浙江大学计算机学院

硕士学位论文

自动化软件测试框架分析及应用

姓名：陆栋

申请学位级别：硕士

专业：计算机科学与技术

指导教师：周波

20080515

摘要

随着计算机应用日益普及和深化，现代软件的规模越来越庞大。以前用手工作坊式方法开发出来的许多大型软件，由于没有进行软件质量管理，因此几乎无法维护，致使项目报废，造成大量人力、物力浪费。如何提高软件质量，保证软件安全性是一个涉及面广、难度很大的课题。超高质量软件的开发技术将是打开21 世纪高技术市场的钥匙，而软件测试则是软件质量保证中的关键技术。一个大型应用软件系统在开发过程中以及研制完成后，如何检验系统是否可靠、是否达到预期目标，成为人们越来越关注的问题。

软件测试一般分为手工测试和自动化测试，我国在软件测试方面起步较晚，并且主要在项目组内部进行手工测试，近几年才逐渐对软件自动化测试引起重视。但如何进行软件测试，特别是如何进行自动化软件测试，则是项目管理层和软件测试人员急需了解的内容[1]。因此，如何构造并建立一个软件自动测试机制，是一个很有实际意义的研究课题。本文针对当今世界最先进的几类软件自动化测试框架进行了细致和深入的分析，并且针对软件自动测试技术中的回归测试构造了自动测试环境，通过规定测试用例的书写方式以及编写关键字驱动脚本，实现了某类金融软件的自动化回归测试，提高了软件测试的效率。

本文通过这个成功案例阐明了在金融及相关软件行业实施自动化测试软件系统的必要性和可行性。并提出了在目前流行的自动化测试框架系统基础上，结合关键字智能化提取改进原有系统框架依赖性过强的想法。并给出了初步的结合持续集成技术的系统自动化整合测试设计。

关键词 自动化测试，框架，关键字驱动

Abstract

As the application of computer technology being used more popular and profound along with the time going, the scale of modern software is becoming huger. For most of the large-scale software in past time using hand made style, these ancient projects were very hard to maintain due to no quality assurance management applied. And caused a lot of waste on projects, they fail and no profits return on investment. How to improve the quality of software, and how to make sure the delivery of software is on schedule is a tough issue. And it's the key to open the hi-tech market of 21st century, the key technology is the software testing. How to validate if the system is reliable and fits the requirement is a more concerned issue, while a huge software application is being built and after delivered.

There are two types of testing, one is manual test, and the other is automation test. Most of the testing is done by manually, and automation test is recognized as a more powerful way in addition to manual nowadays. However, how to execute software testing, especially automation is a more urgent task project manager and software tester should fully understand. Hence, how to build and create a software testing framework is a very realistic research topic. This paper analyzes most of the progressive automation tool in the world with details. And create a regression test over automation technology, with manually written test cases and key-word driven technology, and implement certain types of financial software regression testing.

This paper is based on large-scale automatic application used in financial industry. Author designed and analyzed a certain fund management automation test system, based on popular software testing frameworks and intelligent keywords collection system. Author partly design of the system overall framework, achieve many of these functional modules. Through the success of this case of automatic regression test, this

paper states the need and the feasibility of automatic testing in financial industry. Author also includes the keyword capture technology to enhance the functions of the weakness of system reliability. It gives a preliminary continues integration (CI) technology to the overall system design.

Keywords Software Automation Test, Framework, Key-word Driven

图目录

表目录

浙江大学研究生学位论文独创性声明

本人声明所呈交的学位论文是本人在导师指导下进行的研究工作及取得的研究成果。除了文中特别加以标注和致谢的地方外，论文中不包含其他人已经发表或撰写过的研究成果，也不包含为获得浙江大学或其他教育机构的学位或证书而使用过的材料。与我一同工作的同志对本研究所做的任何贡献均已在论文中作了明确的说明并表示谢意。

学位论文作者签名：　　签字日期：　　年　　月　　日

学位论文版权使用授权书

本学位论文作者完全了解 浙江大学 有权保留并向国家有关部门或机构送交本论文的复印件和磁盘，允许论文被查阅和借阅。本人授权 浙江大学 可以将学位论文的全部或部分内容编入有关数据库进行检索和传播，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。

（保密的学位论文在解密后适用本授权书）

学位论文作者签名：　　导师签名：

签字日期：　　年　　月　　日　　签字日期：2018年 6月6日

致谢

首先要感谢我的导师——周波老师。在我读研期间，他给了我进入道富技术中心实习的机会。这里有非常好的研究和实践的环境。正是在这个技术中心里，我参与研究了本文中提到的自动化软件测试系统。周老师给予我很多有益的指点和帮助。他渊博的知识极大的开阔了我的视野，他诚恳严谨的处事态度给了我很多启发。

研究工作是我们整个开发团队集体智慧的结晶。感谢在这几年里和我一起精诚合作的各位师兄师姐师弟师妹。其中，金一如和罗进开分别是我在不同项目组时的组长，他们都给我的学习提供了很多直接的指导。

同时我也要感谢道富技术中心的杨小虎，孙建伶，李善平这几位老师。他们虽然不是我的直接导师，但同样在我的学习生活上给提供了很多帮助。

最后，非常感谢答辩组的老师能够在百忙之中抽空给予我指导和帮助。

博学笃志，切问近思，这是激励我不断前行的信念。两年的研究生求学生涯感受颇多，我深切地体会到自己所学实乃沧海一粟，同时也明白，惟有在今后的工作中踏实苦干，才不负国家和社会对我的培养。

陆 栋

2008 年 5 月 15 日

第1章 绪论

1.1 研究背景

一九八三年IEEE提出的软件工程标准术语中给软件测试下的定义是:“使用人工或自动手段来运行或测定某个系统的过程,其目的在于检验它是否满足规定的需求或是弄清预期结果与实际结果之间的差别[2]。”这一定义非常明确地提出软件测试以检验是否满足需求为目标。为了达到这一目标,测试活动可采取不同的策略,如随机测试或穷举测试、白盒测试或黑盒测试等。根据软件测试自动化的程度(即对计算机的依赖程度和利用程度),测试活动还可以分为手动测试和自动测试两类。由于自动测试的可重用性和测试的一致性较好,测试执行效率和资源利用率较高,从而使测试过程变得更加系统化、规范化,能够被更迅速、更准确地完成。尤其是对于需重复测试的大规模软件,自动测试是更高效的测试方法,因此软件自动测试技术在软件工程中应用越来越广泛。

在信息时代的今天,面对日益激烈的金融竞争环境,软件行业要为客户提供一流的服务,仅仅使用简单的手工测试这种方式显然不符合客户对金融行业的要求。自动化测试框架系统就是针对手工测试效率低,覆盖面不全等问题产生的。所谓自动化测试系统,简单来说就是用计算机按照特定的流程或商业逻辑来模拟用户执行的操作,比较软件执行的结果与预期的差别,将执行的结果记录到日志文件中去的过程,由于计算机能够做到执行重复的操作,并且每一次的操作都是无差异性的,从而弥补了手工测试对单一重复操作的不足,补强了过去纯手工的软件测试过程(本文将在后面的章节对自动化测试的概念有更加详细的解释)。自动化测试是测试行业智能化中很重要的一个体现,这个转变大大提高了工作效率。

当前,全世界的各大金融软件行业也都开始致力于对企业内各测试部门业务的自动化改造。很多的自动化测试系统开始被软件质量保证人员使用并给用户带来了产品质量的提升。笔者有幸参与了一个为某金融托管银行下的一个基金交易部门开发的基金交易平台的自动化软件测试系统的开发。通过开发实践,笔者更加深刻的理解了金融行业对自动化测试的具体需求,以及如何进行符合金融行业

需要的自动化软件测试框架系统的开发。

1.2 主要工作

前面所提到的基金交易平台系统是一个已投入商用的银行外部交易系统。笔者的主要研究开发工作有：参与了项目的整体系统性测试，在实践的基础上开发了适合项目的自动化测试框架，参与开发了框架中的冒烟测试和回归测试的部分脚本，自动化了大部分文件上传功能的测试脚本，并且对上述的脚本进行了多个开发版本的维护工作。

本框架的核心是一个基于关键字驱动的自动化回归测试系统，这部分也是笔者研究工作的重点。笔者设计并实现了关键字和测试用例的智能化关联，充分利用关键字驱动和数据驱动的优势开发文件上传脚本，实现了脚本测试对象与实际用户界面接口的分离和灵活关联功能。根据具体的商业需求，自动重复执行多个测试场景，同时按照特定的规则记录日志，最后生成测试报告，极大的丰富了测试手段。

1.3 组织结构

本文共分为七章，各章内容如下：

第一章是本文的绪论。本章引入测试自动化的概念，提出了在软件行业实现自动化测试框架的研究课题。并阐述了本文的主要工作。

第二章首先介绍了自动化测试技术的概念和主要的发展阶段。阐明了不同类型自动化测试在金融行业实现的各种好处以及各自的优缺点。自动化测试是软件测试智能化的一部分，这是未来的发展趋势。并提出了一个融合大多数自动化软件测试技术的框架雏形。

第三章深入分析了前一章所提到的几种自动化技术。针对模块化自动化框架，数据驱动自动化框架，以及关键字驱动自动化框架进行了具体分析。并且针对实际应用，进行了细致的介绍。

第四章是重点比较了几种主要的自动化框架的设计思想。重点针对当前主流框架技术的主要优缺点做了细致深入的分析。

第五章具体介绍了笔者自己研发的自动化测试框架，并结合具体项目介绍了各种先进的自动化工具的思想是如何融合到笔者的系统中去的。该系统充分利用了插件可扩展化和模块化设计思想，利用数据驱动和关键字驱动技术，提高了该

框架模型的可扩展性。

第六章提出了将该自动化设计框架进一步改进的构想。主要提出了基于模型的驱动系统和智能化测试用例管理等观点。笔者认为这可以提升框架系统的完整性，健壮性，以及可扩展性，可维护性。并且给出了新的结合新技术的自动化测试系统的整体设计。

第七章是本文的总结。本章对全文做了总结，对将来的工作提出了进一步研究的建议。

第2章 自动化测试技术的发展

2.1 软件测试自动化技术的特点

软件自动化测试是一门技术，并且与手工测试技术存在很大区别。许多组织发现自动化测试相比执行一次手工测试的开销大很多。如果希望从自动化测试中获得收益，则需要仔细选择自动化测试的用例。大量实践中的事实表明，自动化的程度与测试的质量是独立的[3]。无论自动执行，还是手工执行测试都不影响测试的有效性和仿效性。同样，无论自动化测试做得如何出色，如果测试本身是失败的，那么测试结果也将毫无意义。自动化测试通常要比手工测试经济得多，其开销只是手工测试的小部分。自动化测试的方法越好，长期使用获得的收益就越大。

为实现高效的自动测试，必须源于优秀的测试软件，这些测试软件要由经验丰富的测试人员精心设计。在此基础上再应用自动化技术实现自动测试，这样可以获得建立及维护的合理开销。测试自动化开发人员可以不是测试者；也可以不是测试组中的成员[4]。例如，测试组可以由具有商业知识而没有软件开发技术的用户测试人员组成。开发者可能需要构造及维护测试组设计的测试用例以实现测试自动化。

任何测试可以是高质的或劣质的，这取决于测试者实现测试的技术优劣。同样，自动化质量也可以是高质的或劣质的。这取决于测试自动化者的自动化技术，包括确定怎样方便地增加新的自动测试用例，如何维护自动测试以及测试自动化最终能提供什么样的效益[5]。

完善的软件测试可以使软件的质量得到很好的保障，在软件测试工作中，许多操作都是重复的、非智力的。这种情况下，计算机最适合代替人类去完成这些任务，它可以通过建立测试脚本来进行软件的自动测试。如对一个软件进行升级或进行错误修改，由于改动了程序必然会造成软件的不稳定性。若为这个软件建立一个自动测试脚本，此时只需要修改脚本中的少许代码，就可以作为升级后的自动测试脚本对该软件进行回归测试。

2.2 自动化测试技术发展介绍

2.2.1 软件自动化测试发展的三个阶段

基于图形界面（GUI）的软件自动化测试框架和工具的发展大致经历了三个阶段（有专家也据此将测试工具分为三代）：

1. 简单的录制/回放：由工具录制并记录操作的过程和数据形成脚本，通过回放来重复人工操作的过程。在这种模式下，数据和脚本混在一起，几乎一个测试用例对应一个脚本，维护成本很高。而且即使界面的简单变化也需要重新录制，脚本可重复使用的效率低。
2. 数据驱动（data driven）的自动化测试：从数据文件读取输入数据，通过变量的参数化，将测试数据传入测试脚本，不同的数据文件对应不同的测试用例。在这种模式下，数据和脚本分离，脚本的利用率、可维护性大大提高，但受界面变化的影响仍然很大。
3. 关键字驱动（keyword driven）的自动化测试：关键字驱动测试是数据驱动测试的一种改进类型，它将测试逻辑按照关键字进行分解，形成数据文件，关键字对应封装的业务逻辑。主要关键字包括三类：被操作对象（Item）、操作（Operation）和值（value），用面向对象形式可将其表现为 Item.Operation(Value)。关键字驱动的主要思想是：脚本与数据分离、界面元素名与测试内部对象名分离、测试描述与具体实现细节分离。

2.2.2 软件自动化测试脚本模型发展的五个阶段

相对应软件自动化测试发展的三个阶段，软件测试自动化脚本的类型经历了五个阶段，从低到高的依次发展层次是：

1. 线性脚本：通过录制直接产生的线性执行的脚本。
2. 结构化的脚本：具有顺序、循环、分支等结构的脚本。
3. 共享的脚本：可以被多个测试用例使用，被其它脚本调用的脚本。
4. 数据驱动的脚本：数据和流程控制分离的脚本，通过读入数据文件来驱动流程进行的脚本。
5. 关键字驱动的脚本：脚本、数据、业务分离，数据和关键字在不同的数据表中，通过关键字来驱动测试业务逻辑。关键字驱动脚本的特点是它看起来更像描述一个测试事例做什么，而不是如何做。

目前，大多数测试工具处于数据驱动到关键字驱动之间的阶段，有些工具厂商已经提出了声称支持关键字驱动的版本。从以上信息可以看到，自动化测试框架和脚本的发展是和软件工程思想的发展一脉相承的。软件开发的模式从面向机器、到面向过程、再到面向对象、面向服务，是一个从底层到高层、从具体到抽象、复用的粒度从细到粗的发展过程。而软件开发中的模块化、层次化、松耦合等思想对自动化测试框架的设计都具有借鉴意义。

2.2.3 测试自动化的五个成熟度级别

在早期的软件开发中，自动化的测试工具只被看作是一种捕获和回放的工具。当前这个迷思仍然在很多测试人员的思想中，而事实上自动化测试已经远不止捕获和回放这么简单了。按照成熟度自动化的测试可以被划分为5个级别[6]。下文分别用表格的方式阐述了各级别的优缺点和适用的情况。

级别 1：捕获和回放

这是使用自动化测试的最低的级别，同时这并不是自动化测试最有用的使用方式。

表 2.1 简单的捕获和回放

好处	
	自动化的测试脚本能够被自动的生成，而不需要有任何的编程知识。
缺点	
	会有大量的测试脚本，同时当需求和应用发生变化时相应的测试脚本也必须被重新录制。
用法	
	当测试的系统不会发生变化时，也就是小规模的自动化。

级别 2：捕获、编辑和回放

在这个级别中，使用自动化的测试工具来捕获要测试的功能。将测试脚本中的任何固定不变的测试数据，比如名字、账号等等，从测试脚本的代码中完全删除，并将他们转换成为变量。

表 2.2 捕获、编辑和回放

好处	
	测试脚本开始变得更加的完善和灵活，并且可以大大的减少脚本的数量和维护的工作。
缺点	
	需要一定的编知识。频繁的变化可能会引起大量冗余的代码，并且变更和维护几乎是不可能的。

用法	
	当进行回归测试时，被测试的应用有很小的变化，比如仅仅是针对计算的代码变化，但是没有关于GUI界面的变化。

能够使用这种技术通过快速的编制一些测试脚本以检验你的想法来探索你的预定的测试设计。在没有任何象需求或者设计模型这样的文档的情况下第一次操作一个产品时和需要获得一系列内部构建版本的稳定性的第一印象时，使用这种技术。通常如果适当的软件配置管理（SCM）与良好的内建设计相结合时，使用级别2的技术已经足够了。

级别 3：编程和回放

这个级别是面对多个构建版本的有效使用测试自动化的第一个级别。要求项目在实际的投资开始显现之前确保团队和客户对项目的安全感。如果没有对测试自动化工具的适当的培训，测试人员将不具备到达这个级别的能力。在自动化测试工具中的所有测试功能都必须被很好的理解，并且要掌握测试脚本语言的知识。

表 2.3 编程和回放

好处	
	适当的设计是必要的，编码的习惯必须是适当的。使用与开发中相同的编码习惯是非常好的，这将开始搭建起测试和开发之间的桥梁。 在项目的早期就可以开始自动化的测试。能够在项目的早期就开始进行测试脚本的设计，与开发人员交并调查他们认为可能会存在问题的区域，确保了开发人员关注在获得能够被测试的方案上。
缺点	
	要求测试人员具有很好的软件技能，包括设计、开发等。
用法	
	大规模的测试套件被开发、执行和维护的专业自动化测试。

级别 3 能够使用自动化测试并构建不同的回归测试（重用已有的自动化测试用例）。根据经验在看到更多切实的回报之前，为了达到这个级别，有大量的工作和影响项目的活动必须被完成。因此快速的建立和证明自动化测试的价值是至关重要的。找到乏味的测试（例如，边缘测试和特定的功能测试用例是首先进行自动化测试的良好候选者），并且创建少量的能够测试一些基本功能（比如，登录和创建用户等）的自动化测试用例。

级别 4：数据驱动的测试

对于自动化测试来说这是一个专业的测试级别。要利用测试工具提供的所有的测试功能。这是一个强大的测试框架，这个测试框架是基于能够根据被测试系

统的变化快速创建一个测试脚本的测试功能库的。维护的成本相对是比较低的。在测试中会使用到大量真实的数据来反复验证功能的正确性。

表 2.4 数据驱动的测试

好处	
	能够维护和使用良好的并且有效的模拟真实生活中数据的测试数据。
缺点	
	软件开发的技能是基础，并且需要访问相关的测试数据。
用法	
	大规模的测试套件被开发、执行和维护的专业自动化测试。

级别 4 要求一些非常良好的测试数据。一个测试人员必须要花费一些时间来识别在哪里收集数据和收集哪些数据。使用现实生产环境中的数据是最有效的方法，从测试中可以得到最好的回报。使用适当的数据将提供通常仅仅在项目的后期才会发现的或者只有到客户可用性测试时才能发现的错误的能力。如果能够通过使用现实的数据运行大量的测试，无疑对自动化测试是很好的帮助。

级别 5：使用关键词的测试自动化

这是自动化测试的最高级别。主要的思想是将测试用例从测试工具中分离出来。 这个级别要求有一个具有高技能测试人员的团队，这些测试人员能够将测试工具的深层次的知识与具备的较深的编程能力结合起来。这个团队负责在测试工具中生成并维护测试的功能性，能够使测试工具从外部的来源，比如 excel 表或者数据库中获得测试用例并执行。这种测试概念最初是由 CMG 开发的。与 CMG 方案相比，其他的可能的开放源码的方案有被 Carl Nagle 和 SAS Institute 开发的 DDE。使用 DDE 的概念，关注点是当在 Excel 表中创建测试用例的时候，如何使用包括被关联的特定动作词语的一些类型的模板。执行的过程是从 Excel 表中读取测试用例，并将测试用例转换成为测试工具能够理解的形式，然后使用不同的测试流程来执行测试。这个概念变得越来越流行，因为测试与用例一起管理是非常有用的。

表 2.5 使用关键词的测试自动化

好处	
	测试用例的设计被从测试工具中分离了出来——关注在设计良好的测试用例上。允许快速的测试用例的执行和基于用例的更好的估计。
缺点	
	需要一个具有工具技能和开发技能的测试团队，以提供并维护测试工程（框架）。
用法	

	专业的测试自动化将技能的使用最优化的结合起来

如果工具不具备使用内建的对象映射的可能性，那么这个方案对于消除与GUI 相关的大部分维护成本是优秀的。在一些组织中已经创建了这种方案，并且其中的一些已经实现了高度的自动化（60%），并且都得到了巨大的回报。如果测试框架是适当的，能够通过使用 excel 来自动生成实际的测试用例。

这个级别对于那些按照正规基础使用用例的组织或者项目来说是非常优秀的。有多少测试用例的估计是需要的，并且当用例数量适当时需要做的工作也是非常简单的。可以集中时间来生成第一个包含主要业务逻辑的“对象映射”的测试用例（主流程）。依据被测试应用的复杂程度，通常这会花费大约半天到一天的时间。后续的每一个测试用例大概会花费 15 到 20 分钟的时间，因为通常多数的测试用例可以复制已有的测试用例，只需要对其进行必要的修改，通常这种修改是有限的。动作词语框架能够通过使用例紧密的并行测试，针对用例开发测试脚本变得可能。

“工欲善其事，必先利其器”，有了合适的工具，还不能立即开始软件自动化测试。原因在于，并不是所有的手工测试都适合作为自动化测试的方案，通常与用例的复杂程度，数据关联的程度，业务需求是否经常变化等因素有关，下一章将就不适合做自动化的情况做一个概述。笔者想说明的问题是，不是所有项目，所有模块都适合做自动化测试的，只有符合特定要求的自动化测试方案才能真正有效提高项目的质量。

2.3 不适合软件测试自动化的情况及分析

2.3.1 当项目在时间上紧迫，并且已经落后了，再使用自动化测试

这种情况发生在真实的项目中，并且笔者曾亲身经历过。由于测试的时间由于项目前期的延迟而被压缩，质量保证项目组想尽办法加快测试速度，提出使用自动化的方法。实际上，正确的思想应该是：由于时间很急迫，所以决不应该使用自动化测试[7]。应该采取的方式是，通过和项目经理以及项目组的沟通，看是否可以将测试用例按照优先级划分，并且提出各层次中的用例如果漏测可能带来的对项目影响，然后决定是否放弃一些用例，而承担漏测的风险；或者增加投入的成本来弥补时间上的不足，但通常这不是一个好的主意。有关这方面的论述由于和本主题无关，因此不展开论述，但是，要说明的问题是，绝对不能期待在时间紧

急的时候再开始自动化测试方案，应该要在项目间隙，或者项目开始的早期就考虑到自动化策略，从而纳入整个项目计划中，并定期做关键点进度的审查。

如果项目已经陷入到了麻烦之中，不应该实施自动化的功能测试。项目很可能因为需要大量的测试框架的准备和实施会被拖垮。应该将重点放在以下的事情上：

- 优化测试的过程。调查并建议在目前工作基础上的测试方法和过程。可以借鉴 RUP 的相关思想和过程。
- 引进或者使得单元/组件测试正式化。这是能够快速获得受益的很好的方法。如果正式的组件测试被使用，可以使用 XUnit 系列工具进行单元或者组件测试。根据经验尽早的使用 XUnit 是非常值得的。在一个引入 XUnit 的项目中，通常会在组件的级别得到百分之三十的效能提升。
- 仅仅在项目团队可以很肯定下列问题时：
  1. 项目能够被适当的推延。
  2. 存在能够通过实施自动化测试被达到的精确的目标。
  3. 项目具备建立适当的测试框架的必要条件。

那么，可以在一个时间紧迫的项目中适当的实施测试自动化。但是根据经验这种情况是很难发生的。

2.3.2 不需要经过培训，就能使用自动化工具

所有的人都在某一些方面具有一定的经验，也许项目没有时间花费在使用新工具的培训上。当一个对自动化工具还不是很熟悉的组织或者项目团队开始实施自动化测试时，培训和指导是至关重要的。如果允许组织或者项目团队在没有关于应该如何做的任何知识的情况下实施自动化的测试，那将肯定会以失败告终。用于实施自动化测试方案的预算会被超出，测试会被延误并且更坏的情况是自动化测试将被放弃。组织和项目团队需要尽量避免一些认识上的缺陷，尤其是自动化测试的维护成本和当测试人员尝试和确认工具如何工作时产生的挫败感。需要确保的是测试过程是适当的，如果测试过程是不合理的，引入自动化测试只会给软件组织或者项目团队带来更大的混乱[8]。因此，希望实施自动化测试方案的组织或者项目团队应该在实施之前建立“训练营”，并将重点放在培训测试团队能够很好的利用一个原型的项目上。

2.3.3 必须要100%的自动化

从管理的角度来说，100%的自动化目标只是一个从理论上可能达到的，但是实际上达到100%的自动化的代价是十分昂贵的。一个40-60%的利用自动化的程度已经是非常好的了。达到这个级别以上将增加测试相关的维护成本。由于对每一个构建版本的需求变化的复杂度，将花费更多的时间在变更测试用例上以使他们能够正确的运行。在这种情况下，通过告知管理层100%的自动化目标是相当昂贵的来确立一个合理的期望值才是明智之举。对于决定自动化一个测试用例的一般规则是这个测试用例必须被运行4次以上。这个数字是基于用户对测试工具有良好的技能并且有一个良好的测试框架的。如果情况不是这样的，整个数字能够是10-20次或者更高。一个例子，在某个项目中测试人员花费两周的时间将手工测试的23天的任务转换成了自动化测试的用例。在完成时，项目能够在4个小时在多个平台上运行相同数量的测试用例。

2.3.4 考虑引入测试框架

测试框架对于产生成功的测试自动化的适当基础是重要的。很多考虑必须被解决以使测试自动化更加有效地被使用[9]。重点必须在：

- **维护成本** 维护成本是成功的使用自动化测试的最重要的问题之一。维护成本直接联系到前面已经提到过的自动化测试的成熟度。组织或者项目必须至少要在成熟度的3级使用高度的测试库才能使维护和更新测试功能变得容易。
- **测试数据** 什么样类型的数据将被使用？要为每一个测试用例生成测试数据还是使用在被测试应用中已有的数据。在很多的情况下一个测试数据被创建了，删除他们是不可能的。
- **可测试性** 自动化测试方案能够有效的测试吗？例如，被适当命名的对象（不仅仅是索引Id）。一个简单的例子是所有的对话框都有相同的 #id 和相同的标题，所不同的仅仅是显示的文字信息。当测试应该覆盖多种语言的方案时，对话框的测试就是一个挑战。
- **测试人员的技能** 被包括在自动化测试的创建中的人员应该具有什么样的技能呢？如果他们具有良好的开发背景，那么成熟度3级是足够了。如果他们有很少的或者根本没有开发的经验，那么被迫使找到或者培训一个自动化测试专家的小组，并直接到达成熟度5级，在成熟度5级测试的创建与实际的

测试执行被分离开。

- **一个好的构建过程** 自动化测试的引入在“构建团队”上加入了一些约束。为了实现自动化测试的高利用率（回归测试），要求具有一个高的构建频率。每周仅仅运行一次自动化的测试不是好的自动化测试的使用率。将回归测试增加到每天一次将帮助快速的发现新的问题并使开发人员更加容易的发现问题的根源，因为对测试的反馈时间是比较短的（开发人员能够记住他们昨天做了什么）。
- **所有权** 不同的测试库的所有权的定义是重要的。一个好的方案会将测试库的组织划分为三个级别：

  级别 1——全局的：这个一个通常的级别。被存储在这个级别的测试功能能够被所有的项目访问。通用的和通常的功能像登录、创建一个用户都是这个级别很好的候选者。

  级别 2——项目：在这个级别的测试功能是与特定的测试项目相关的，但是通常在项目中有用的比一定在项目外是有用的。通常级别 2 是级别 1 的功能的提供者。

  级别 3——脚本：功能被直接关联到特定的测试脚本。 在这个级别中，通常一个测试功能的第一个版本是被开发的。在新的测试脚本的创建期间已有测试功能的重用性被发现，并被移到了级别 2 中。 在这个级别上尽量最小化功能的数量，因为它将增加维护的工作量。

还有很多有关测试框架的问题，但是这里所提及的是一些基本的要被解决的问题。

2.4 在哪里使用自动化测试

有很多的情况下使用自动化的测试可以降低测试成本。表 2.6 列举了在各种不同的测试技术、测试周期中使用自动化测试的技术分析，从而更详细的说明在何种情况下需要自动化，在何种情况下不需要。

表 2.6 测试自动化技术的分类

技术	描述	备注
单元测试/组件测试	这个测试工作通常是开发人员的职责，很多不同的方法能够被使用，比如“测试先行”，它是一个测试框架，开发人员在编写代码前编写不同的	通过使用正式的单元测试，不仅能够帮助开发人员产出更加稳定的代码而且能够使软件的整体质量更加的好。

	单元测试。当测试通过时，代码也被完成了。	
冒烟测试	冒烟测试是一般验证别测试系统的功能性测试用例的集合。冒烟测试背后的思想是确保基础是可以工作的，以便“下一步的”测试工作能够开始。	在构建过程能够确保构建已经为测试准备好时，冒烟测试通常是自动化运行。
功能/集成测试	这里测试的工作关注在验证不同的组件之间的集成上。	这些类型的测试通常是被测试系统更加复杂测试的基础，大量的边缘测试被合并以制造出不同的错误处理测试。
系统测试——用例测试	这种测试是通过执行用户场景模拟真实用户使用系统以证明系统具有被期望的功能的测试。	这里不需要进行自动化的测试。安装测试、安全性测试通常是由手工完成的，因为系统的环境是恒定不变的。
回归测试	回归测试实际上是重复已经存在的测试。通常如果是手工完成的话，这种测试只在项目的结尾执行执行一到两次。	这里完全有潜力应用自动化的测试。在每次构建完成后执行自动化的回归测试，以验证被测试系统的改变是否影响了系统的其他功能。
性能测试	性能测试包括以下不同测试形式： - 负载测试 - 压力测试 - 并发测试	如果没有自动化的测试工具，几乎不可能执行通过模拟用户负载实现的高密集度的性能测试。

2.5 什么时候使用自动化测试

在上一节的基础上，进一步分析自动化测试接入的时机，对测试人员有目的的在项目进行中选择时机实行自动化测试有着十分积极的意义。表2.7对什么时候应该使用自动化测试和什么时候应该使用手工测试进行了一个概要的总结：

表 2.7 使用测试自动化的条件

使用自动化测试	使用手工测试

• 项目没有严格的时间压力
• 具有良好定义的测试策略和测试计划
  ○ 知道要测试什么
  ○ 知道什么时候测试
• 对于自动化测试拥有一个确定的优秀测试框架和优秀候选者
• 能够确保多个测试运行的构建策略
• 多平台环境需要被测试
• 拥有运行测试的硬件
	• 没有适当的测试过程
	• 没有一个测试什么，什么时候测试的清晰的蓝图
	• 在一个项目中，都是新人，并且还不是完全的理解方案的功能性或者设计
	• 整个项目在时间的压力下
	• 在团队中没有资源或者具有自动化测试技能的人
	• 没有硬件

• 拥有关注在自动化过程上的资源 • 被测试系统是可自动化测试的	

对于从事自动化测试的人员，一定要关注将自动化测试与手工测试结合起来使用。首先，对于自动化测试率的目标是 10/90 （10%的自动化测试和 90%的手工测试）。当这些目标都实现了，可以将自动化测试的使用率提高。创建自动化测试的测试用例要比创建手工测试的测试用例花费更多的时间，因此不要将所有的测试时间都用在自动化的测试用例上。同时要在测试期间对每一个发现的错误花费一定的时间去处理，将缺失的测试用例及时补上，经验表明，找到错误的地方往往存在着更大的可能性发现新的错误。

2.6 自动化测试的好处

如果在组织中引入自动化测试，有很多不同的方面被包含了进去。今天在测试工作如何被进行上有很多不同的观点，为了能够成功的实施自动化测试，笔者列举了应该优先考虑的一些问题：

- 测试需要覆盖什么？什么是现有手工测试没有覆盖的？
- 由于遗漏的测试，没有发现的‘bug’会带来什么样的成本？
- 由于不好的测试，破坏已有功能性的成本是多少？
- 如果‘琐碎的’测试被每天运行，对于项目意味着什么？
- 如果 QA 能够每天向开发人员提供他们最近代码变更相关的反馈，对项目有怎样的影响？

这些问题都能够适当的使用自动化测试来满足。根据笔者的经验快速的向开发人员反馈并每天运行测试对于达到项目质量更高的水平是非常有好处的。自动化测试同时对项目有以下的贡献：

- 降低风险 — 知道测试了什么和没测试什么
- 测试能在项目的早期开始并随着时间一直扩展
- 快速的反馈 — 自动化测试用例能够随时的运行
- 在多个平台上的测试能够同时进行
- 更好的估计 — 能够对测试进度和使用的时间有更好的了解和预测
- 优秀人员的集中 — 能够得到一个专家的团队，并将他们的知识传播给其

他的项目

2.7 本章小结

本章首先介绍了自动化测试技术发展的规律和特点。

然后，介绍了自动化测试技术的发展过程，分为测试框架发展的三个阶段和测试脚本结构发展的五个阶段描述。以及用成熟度模型来衡量的五个不同的自动化级别。

之后列举了一些在软件行业中实施自动化测试一些错误认识，引出下文将深入分析的几种主流自动化测试的框架，以及分别对应的优点和不足。

最后介绍了在实际项目中，应该什么时候使用自动化的时机和条件，以及自动化测试能给项目带来的好处。

第3章 自动化测试框架概述及原理分析

3.1 自动化测试框架概述

自动化测试框架是一套集假设，概念和实践为一体，提供自动化软件测试支持的集合体。仅仅使用录制和回放测试用例的工具，例如 IBM Rational Robot，有着种种的缺陷。仅使用捕捉工具运行复杂的和强大的测试耗费时间并且昂贵。因为这些测试仅仅实现简单验证功能，测试用例的功能性难以跟踪和重现，维护费用也过高。一个更好的选择是让刚刚开始组建的测试团队使用一套自动化测试框架，它集成了假设，概念和实践，组成一个工作平台支持自动化测试。本章中，笔者尝试去解释一些所用过的或接触过的测试框架：模块化测试脚本，测试库结构，数据驱动，关键字驱动/表格驱动，以及混合测试自动化。单纯地比较孰优孰劣并不是笔者的本意，目的是尽量真实了解每一种框架的优劣，以及对真实项目的价值。

3.2 测试脚本模块化框架

模块化脚本框架要求创建一系列小的，独立的脚本，可以展现 AUT（被测应用程序）的模块，层次以及功能。这些小的脚本按照一定的结构按层次组合成一些更大的测试，从而实现一些复杂的测试流程（test scenarios）。在所有的框架中，这种是最简单，最容易掌握的。它源自一种著名的编程模式——代理模式（Proxy），创建一个抽象层在模块的外层，从而将真实的模块隐藏起来[10]。测试脚本用类似的方法来管理的目的是加强测试自动化脚本的可维护性和可扩展性。为了演示这种框架，拿 Windows 的计算器来自动化一个简单的测试用例，用来测试基本的功能（加减乘除）。

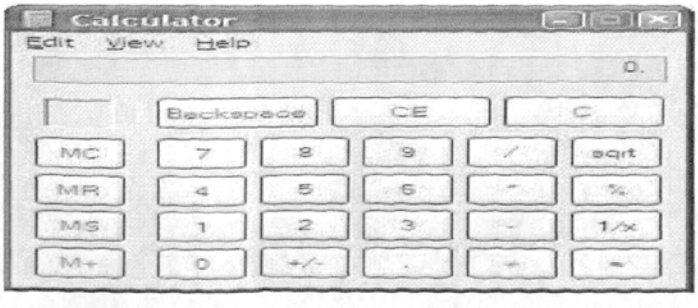

图 3.1 Window 计算器界面图

在脚本的最底层，是独立的测试脚本，分别测试加减乘除。例如下面两个脚本将演示加法和减法的使用：

```
Sub Main
    Window Set Context, "Caption=Calculator", ""
    '5
    PushButton Click, "ObjectIndex=10"
    '+
    PushButton Click, "ObjectIndex=20"
    '6
    PushButton Click, "ObjectIndex=14"
    '=
    PushButton Click, "ObjectIndex=21"
    '11
    Result = LabelUP (CompareProperties, "Text=11.", "UP=Object Properties")
End Sub

Sub Main
    Window Set Context, "Caption=Calculator", ""
    '20
    PushButton Click, "ObjectIndex=11"
    PushButton Click, "ObjectIndex=8"
    '-
    PushButton Click, "ObjectIndex=19"
    '10
    PushButton Click, "ObjectIndex=7"
    PushButton Click, "ObjectIndex=8"
    '=
    PushButton Click, "ObjectIndex=21"
    '10
    Result = LabelUP (CompareProperties, "Text=10.", "UP=Object Properties")
End Sub
```

接下来两个脚本将会用来展示标准视图和科学计算视图的差别。会在标准视图脚本中调用之前的脚本。

```
'Test Script Modularity Framework
'Script for Standard View

Sub Main
    'Test Add Functionality
    CallScript "Test Script Mod Framework - Add"

    'Test Subtract Functionality
```

```
    CallScript "Test Script Mod Framework - Substract"

    'Test Divide Functionality
    CallScript "Test Script Mod Framework - Divide"

    'Test Multiply Functionality
    CallScript "Test Script Mod Framework - Multiply"

End Sub
```

最后，也就是这个案例的最高层测试用例将会包括不同视图功能的测试：

```
'Test Script Modularity Framework
'Top level script - represents test case

Sub Main
    'Test the Standard View
    CallScript "Test Script Mod Framework - Standard"

    'Test the Scientific View
    CallScript "Test Script Mod Framework - Scientific"

End Sub
```

从这个简单的例子中可以看到，这个框架产生了高度的模块化层次结构，提高了整体测试用例的可维护性。如果一个控件从原来的位置上发生了变动，所需要做的就是从底层调用这个控件的脚本，而不是所有与这个控件有关的测试用例。

3.3 测试库框架

测试库框架与测试脚本模块化很类似，它能提供同样的好处，但是它把 AUT 分解成了一些过程和功能模块而不是脚本。这个框架要求创建代表 AUT 模块，功能，层次的库文件（SQA 基础库，APIs，DLLs 等等）。这些库文件可以被直接从测试脚本调用。为了展示这个框架，用 SQA 基础库来自动化与上面同样的测试用例。这个库包含了操作基本动作的功能。以下举例的是库文件和源文件。

```
'Header File
'Test Library Architecture Framework
"Functions Library

Declare Sub StandardViewFunction BasicLib "Functions Library" (OperandOne As
Integer, _OperandTwo As Interger, _Operation As String)

'Library Source File
'Test Library Architecture Framework
```

```
'Functions Library

Sub StandardViewFunction (OperandOne As Integer, _OperandTwo As
Interger, _Operation As String)
    'Click on first operand
    Select Case OperandOne
        Case 0
            PushButton Click, "ObjectIndex=8"
        Case 1
            PushButton Click, "ObjectIndex=7"
        Case 2
            PushButton Click, "ObjectIndex=11"
        Case 3
            PushButton Click, "ObjectIndex=15"
        Case 4
            PushButton Click, "ObjectIndex=6"
        Case 5
            PushButton Click, "ObjectIndex=10"
        Case 6
            PushButton Click, "ObjectIndex=14"
        Case 7
            PushButton Click, "ObjectIndex=5"
        Case 8
            PushButton Click, "ObjectIndex=9"
        Case 9
            PushButton Click, "ObjectIndex=13"
     End Select

    'Click on first operand
    Select Case OperandOne
        Case "+"
            PushButton Click, "ObjectIndex=8"
        Case "-"
            PushButton Click, "ObjectIndex=7"
        Case "*"
            PushButton Click, "ObjectIndex=11"
        Case "/"
            PushButton Click, "ObjectIndex=15"
     End Select

    'Click on first operand
    Select Case OperandOne
        Case 0
            PushButton Click, "ObjectIndex=8"
```

```
        Case 0
            PushButton Click, "ObjectIndex=7"
        Case 0
            PushButton Click, "ObjectIndex=11"
        Case 0
            PushButton Click, "ObjectIndex=15"
        Case 0
            PushButton Click, "ObjectIndex=6"
        Case 0
            PushButton Click, "ObjectIndex=10"
        Case 0
            PushButton Click, "ObjectIndex=14"
        Case 0
            PushButton Click, "ObjectIndex=5"
        Case 0
            PushButton Click, "ObjectIndex=9"
        Case 0
            PushButton Click, "ObjectIndex=13"
    End Select
    '=
    PushButton Click, "ObjectIndex=21"

End Sub
```

使用这个库，接下来的测试用例就能调用这些基础操作了。

```
'Test Library Architecture Framework
'Test Case script

'$Include "Functions Library.sbh"
Sub Main
    'Test the Standard View
    Window Set Context, "Caption=Calculator", ""

    'Test Add Functionalty
    StandardViewFunction 3,4,"+"
    Result = LabelVP (CompareProperties, "Text=7.", "VP=Add")

    'Test Subtract Functionalty
    StandardViewFunction 3,2,"-"
    Result = LabelVP (CompareProperties, "Text=1.", "VP=Sub")

    'Test Multiply Functionalty
    StandardViewFunction 4,2,"*"
    Result = LabelVP (CompareProperties, "Text=8.", "VP=Mult")
```

```
    'Test Divide Functionalty
    StandardViewFunction 10,5,"/"
    Result = LabelVP (CompareProperties, "Text=2.", "VP=Div")

End Sub
```

从这个例子中不难发现，使用测试脚本库的框架同样产生了高度的模块化的同时增加了整个测试套件的可维护性。就像在模块化的测试脚本中，如果一个控件从原来的位置移动了，只需要改动那个库文件，而不需要改动所有调用这个库文件的测试脚本，隔离了GUI的变化和测试流程（业务逻辑），使得测试脚本的可维护性大大增强了。

3.4 数据驱动测试框架

数据驱动测试是一种从数据文件（数据池，ODBC源，CSV文件，Excel文件，DAO对象，ADO对象，等等）中读取输入和输出值，然后将这些值装载进录制或者手动编码的脚本的测试框架。在这个框架中，变量同时存在于输入和输出值中。在程序运行时，读取数据，记录测试状态和信息，都是在测试脚本中编码的。

就测试用例是存储在数据文件中而不是在脚本中这点，数据驱动和表格驱动采用了相似的方法；对数据来说，脚本只不过是一个驱动，或者传递机制。但是和表格驱动所不同的是，驱动数据并没有存在数据表格里。数据驱动的测试，只有测试数据存放在数据文件中[11]。IBM Rational 工具集有原始的数据驱动功能，由SQABasic语言和IBM Rational数据池（Datapool)结构支持实现。为了展示这个框架的使用，以测试Rational中经典程序A范例程序的下单功能为例。(参见图3.2）

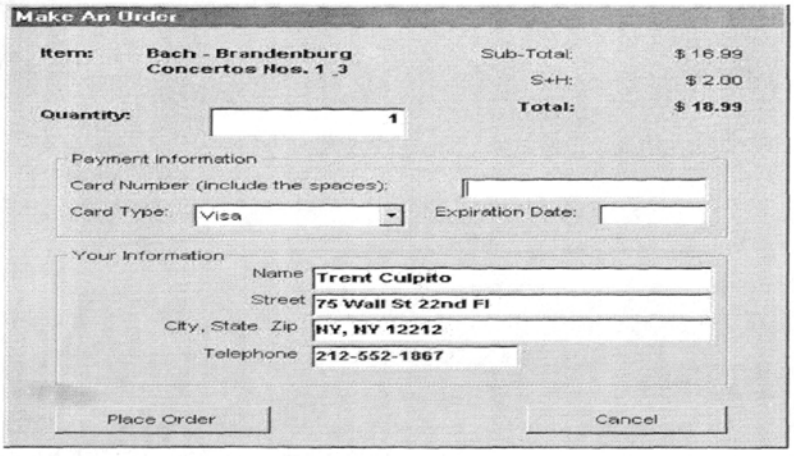

图 3.2 Rational Classic A 订单界面

如果记录下这个窗口中的数据操作，得到如下脚本：

```
'Data Driven Framework
'Test Case Script

Sub Main
    'Make An Order
    Window Set Context, "Name=frmOrder", ""

    'Card Number
    EditBox Click, "Name=txtCreditCard", "Coords=16,9"
    InputKeys "3333444455556666"

    'Expiration Date
    EditBox Click, "Name=txtExpirationDate", "Coords=6,7"
    InputKeys "3333444455556666"

    'Place Order
    PushButton Click, "Name=cmdOrder"

    'Confirmation Screen
    Window SetContext, "Name=frmConfirm", ""
    PushButton Click, "Name=cmdOK"

End Sub
```

可以只用数据池来创建测试用例，用来测试正确的和不正确的信用卡号码和过期时间。数据池如图 3.3，用来测试数据域的用例。

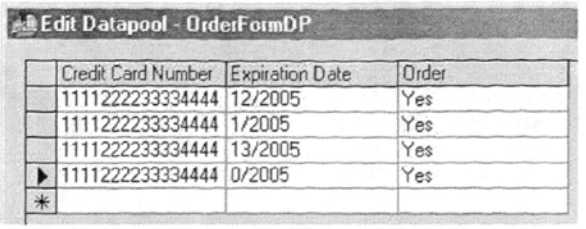

图 3.3 用来测试数据区域的数据池范例

如果修改脚本来接受数据池中的数据，就得到如下脚本：

```
'Data Driven Framework
'Test Case Script

'$Include "SQAUTIL.SBH"

Sub Main
    Dim Result As Integer
    Dim DatapoolHandle As Long
    Dim DatapoolReturnValue As Variant

    'Open the datapool
    DatapoolHandle = SQADatapoolOpen("OrderFormDP")
    '...Add error checking....

    'Loop through the datapool
    While SQADatapoolFetch(DatapoolHandle) = dqaDpSuccess
        'Open Order Form
        Window SetContext, "Name=frmMain", ""
        PushButton Click, "Name=cmdOrder"
        Window SetContext, "Name=frmOrder", ""

        'Card Number
        Result = SQADatapoolValue(DatapoolHanle, "Credit Card Number",
DatapoolReturnValue)
        "...Add error checking....
        EditBox Click, "Name=txtCreditCard", "Coords=16,9"
        '...Clear Value....
        InputKeys DatapoolReturnValue

        'Expiration Date
        Result = SQADatapoolValue(DatapoolHandle, "Expiration Date",
DatapoolReturnValue)
        '...Add error checking....
        '...Clear Value...
```

```
        EditBox Click, "Name=txtExpirationDate", "Coords=6,7"
        InputKeys DatapoolReturnValue

        'Place Order
        Result = SQADatapoolValue(DatapoolHandle, "Order",
DatapoolReturnValue)
        If UCASE(DatapoolReturnValue) = "YES" Then
            PushButton Click, "Name=cmdOrder"

            'Confirmation Screen
            Window SetContext, "Name=frmConfirm", ""
            Pushbutton Click, "Name=cmdOK"
        Else
            PushButton Click, "Name=cmdCancel"
        End If
        Wend 'Go fetch next row

        'Close datapool
        Result = SQADatapoolClose(DatapoolHandle)
        '...Add error checking....
End Sub
```

这个框架尝试减少执行所有测试数据所需要撰写的脚本数量，并且提供了更好的灵活性，尤其当需要开发对缺陷的重复验证以及执行维护功能。和表格驱动类似的，数据驱动只要非常少的代码，就能覆盖大量的测试用例。这个框架由于其灵活性和成熟性，有很多工具和文档都支持它，可以说是当前自动化测试框架技术中用的最多、最普遍的技术之一。

3.5 关键字驱动（表格驱动）框架

关键字驱动和表格驱动测试指的是同样一个事物，就是独立与应用程序的自动化框架。这个框架需要有数据表格和关键字开发的支持，与执行这些测试脚本驱动 AUT 执行的测试代码隔离。关键字驱动测试看起来与手工测试很相像。在一个关键字驱动的测试中，AUT 的功能放在一个类似表格的文档里，每一行就是一个执行步骤[12]。

如果需要将自己在计算器软件上用鼠标点选操作的步骤和关键字驱动表格对应起来，就有了如下的表格。Window 栏包含应用程序窗口的名称。Control 栏名称对应鼠标点选的控件名。Action 栏列出了鼠标的操作。Arguments 栏区分了不同的控件名称（数字，加减符号等）。

Window	Control	Action	Arguments
Calculator	Menu	View, Standard	
Calculator	Pushbutton	Click	1
Calculator	Pushbutton	Click	+
Calculator	Pushbutton	Click	3
Calculator	Pushbutton	Click	=
Calculator	Verify Result	4	
Calculator	Clear		
Calculator	Pushbutton	Click	6
Calculator	Pushbutton	Click	-
Calculator	Pushbutton	Click	3
Calculator	Pushbutton	Click	=
Calculator	Verify Result	3	

这个表格演示了一个完整的测试；为了展示一系列的操作还需要更多的用例。一旦创建了数据表格，只需要简单的写一个程序或者一组脚本来读每一行数据，基于 Action 域的关键字来执行每一步，执行错误检查，写日志文件以及相关的操作都能轻易的完成。这个程序看起来和下面的伪代码类似：

```
Main Script / Program
Connect to data tables.
Read in row and parse out values.
Pass values to appropriate functions.
Close connection to data tables.
Menu Module
Set focus to window.
Select the menu pad option.
Return.
Pushbutton Module
Set focus to window.
Push the button based on argument.
Return.
Verify Result Module
Set focus to window.
Get contents from label.
Compare contents with argument value.
Log results.
Return.
```

从这个例子中，可以看到这个框架只需要很少的代码就能产生很多测试用例。目前支持数据驱动的大部分软件都能经过改造支持关键字驱动[13]，例如 IBM Rational 有支持这种交互性的文件读取工具，其他一些开源工具如，ThoughtWorks 的 Selenium 和 FIT 在设计的时候就是利用关键字脚本驱动的，因此从设计思想

和架构上完全支持表格驱动的所有功能，下文就来研究这两个纯粹由关键字驱动的测试工具，从中能发现和体会到很多帮助笔者搭建自己的自动化框架的优秀设计思想。

3.5.1 Selenium 中的关键字驱动

Selenium 是 ThoughtWorks 专门为 Web 应用程序编写的一个验收测试工具。据 Selenium 主页所说，与其他测试工具相比，使用 Selenium 的最大好处是：

“能够直接在浏览器中运行，就像真实用户所做的一样。Selenium 测试可以在 Windows、Linux 和 Macintosh 上的 Internet Explorer、Mozilla 和 Firefox 中运行。其他测试工具都不能覆盖如此多的平台。”[14]

使用 Selenium 和在浏览器中运行测试还有很多其他好处。下面是主要的两大好处：

- 通过编写模仿用户操作的 Selenium 测试脚本，可以从终端用户的角度来测试应用程序。
- 通过在不同浏览器中运行测试，更容易发现浏览器的不兼容性。

Selenium 的核心，也称 browser bot，是用 JavaScript 编写的。这使得测试脚本可以在受支持的浏览器中运行。browser bot 负责执行从测试脚本接收到的命令，测试脚本要么是用 HTML 的表布局编写的，要么是使用一种受支持的编程语言编写的。

Selenium 适用于以下浏览器：

	Internet Explorer	Mozilla	Firefox	Safari
Windows XP	6.0	1.6+, 1.7+	0.8+, 0.9+, 1.0	
Red Hat Linux		1.6+, 1.7+	0.8+, 0.9+, 1.0+	
Mac OS X 10.3	*不支持*	1.6+, 1.7+	0.8+, 0.9+, 1.0+	1.3+

Selenium 命令相当于关键字驱动中的 Action 字段关键字集合。通过 Selenium 命令，脚本编写者可以描述 browser bot 在浏览器中所执行的操作。可以将这些命令分成两类 —— 操作（action）和断言（assertion）：操作模拟用户与 Web 应用程序的交互。例如，单击一个按钮和填写一个表单，这些都是常见的用户操作，可以用 Selenium 命令来自动化这些操作。断言验证一个命令的预期结果。常见

的断言包括验证页面内容或当前位置是否正确。

Selenium 模式反映了两种不同层次的测试驱动方式。两种模式分别是：test runner 和 driven。Test runner 模式简单的将录制的操作转换成关键字表格，通过在浏览器里回放进行回归测试。Driven 模式通过编写测试脚本来提供跟多的灵活性。虽然脚本的编写往往要更复杂一些，因为它们是用编程语言编写的。但是如果使用 Python 或 Ruby 之类的高级动态编程语言，那么这种复杂性方面的差异就很小。两种模式之间最大的不同点在于，如果使用 driven 脚本，测试有一部分在浏览器之外运行，而如果使用 test runner 脚本，测试是完全在浏览器中运行的。

Test runner 模式 Selenium test runner 脚本，也称测试用例（test case），是用 HTML 语言通过一个简单的表布局编写的，如表 3.1 所示。

表 3.1 Selenium 测试用例的结构

```
<table border="1">
<tr>
<td>First command</td>
<td>Target</td>
<td>Value</td>
</tr>
<tr>
<td>Second command</td>
<td>Target</td>
<td>Value</td>
</tr>
</table>
```

Test runner 脚本通常与所测试的应用程序（AUT）部署在同一个服务器上。这是因为 browser bot 使用 JavaScript 来模拟用户操作。test runner 脚本使用与 xUnit 框架相同的测试套件（test suite）和测试用例概念。测试用例和命令按照它们在测试套件和测试用例中出现的顺序依次执行。在表 3.1 中：

第一列包含命令或断言。第二列包含命令或断言的目标（target）。这里可以用多种受支持的组件定位符中的一种来指定目标。通常使用的是组件的 ID 或名称，但 XPath 和 DOM 定位符也是受支持的。通常使用这种方式定位可以获得更好的灵活性，这一点将在第 5 章介绍笔者自己设计的工具时介绍，并且与 Rational 的专利技术做一对比。第三列包含用于为命令或断言指定参数的值。例如，当使用 type 命令时，这一列可能就是一个文本域所期望的值。

即使对于非技术人员来说，test runner 脚本也易于阅读和编写。当在一个浏览器

中打开表 3.1 中的例子时，看到是是一个典型的关键字驱动的表格：

First command	Target	Value
Second command	Target	Value

Driven 模式 Driven 脚本是用多种受支持的编程语言中的一种编写的——目前可用的有 Java、Ruby 和 Python 驱动程序。这些脚本在浏览器之外的一个单独的进程中运行。驱动程序的任务是执行测试脚本，并通过与运行在浏览器中的 browser bot 进行通信来驱动浏览器。驱动程序与 browser bot 之间的通信使用一种简单的特定于 Selenium 的连接语言 Selenese。

Driven 脚本比 Test runner 脚本更强大、更灵活，可以将它们与 xUnit 框架集成。Driven 脚本的缺点（与 Test runner 脚本相比）是，这种脚本编写和部署起来更复杂。driven 脚本更依赖于应用程序运行时环境。例如，Java 驱动程序使用一个嵌入式 Jetty 或 Tomcat 实例来部署所测试的应用程序。

3.5.2 FIT 中的关键字驱动

FIT（Framework for Integrated Tests） 是一种通用的开放框架[15]，是由 Ward Cunningham 开发的，可以帮助开发人员进行自动化的确认测试。回归确认测试是轻型开发模式（XP、Crystal 等）测试活动的另一个优秀思路也是采取轻型开发模式的必要条件之一。在只有测试实现了自动化，回归测试才能实现，重构（采取轻型开发模式另外的一个必要条件）才能够贯彻，而迭代也才能够进行。FIT 利用 JUnit 并扩展了 JUnit 的测试功能。

长期以来，在软件开发中一直存在着两个主要问题：

第一，业务如何通过应用程序与其所需内容通信；

第二，工程师如何验证他们是否正在构建满足业务需要的正确软件。

多年来，为了解决这些关心的问题，已探索了许多方法和框架，但直到出现 Framework for Integrated Tests (FIT) 以后，才找到了解决这些问题的简便而直观的方法。

使用 FIT 可以编写出可以自动运行的确认测试用例，可以用来确认开发出来的软件是否满足了用户所需的功能，可以作为持续构建过程的一部分来确保所构建出来的版本是正确的。但是，FIT 还有另外一个更为重要的功能，那就是在软

件开发中增强协作，尤其是开发团队和客户、领域专家之间的协作。这种协作可以有效地降低软件开发中的不必要的复杂性，加速反馈，并确保最大程度地为客户提供最高的价值。

简单来讲，FIT 就是一个关键字驱动的测试软件，它能够读取 HTML 文件中的表格（这些表格可以通过 Microsoft Word 或者 Excel 产生）。针对每个表格，都会由一个程序员编写的"fixture"（装置）来解释。该 fixture 会驱动"被测系统 (System Under Test)"来对表格中给出的测试用例进行检验。

Fixture 充当 Fit 表格和被测试系统间的媒介，起协调作用，完成表格中给出的测试。FIT 中提供了好几种类型的 Fixture，它们分别用于处理不同的情形。Fixture 的形式有 3 种：

ColumnFixture（对应于"列"表），"列"表的形式如下表所示：

计算奖学金方法

Score	Scholarship()
1000	0
1999	0
2000	500
2050	500
2100	1000
2200	1500
2300	2000
2350	2000
2400	2500

RowFixture（对应于"行"表），"行"表的形式如下图所示：

打折数据列表

Order	future value	max owing	min purchase	discount percent
1	low	0.00	0.00	0
2	low	0.00	2000.00	3
3	medium	500.00	600.00	3
4	medium	0.00	500.00	5
5	high	2000.00	2000.00	10

ActionFixture（对应为"行为"表），表明以表格给出的测试用例的一系列的操作步骤。见下表。

fit.ActionFixture
start cstc.fitexam.coffeemaker.AddInventory
enter units coffee 3
enter units milk 5

enter units sugar 6 enter units chocolate 7 check coffee inventory 18 check milk inventory 20 check sugar inventory 21 check chocolate inventory 22

在上表中，第一列给出了执行的命令，这里共有3个命令，但是其它的命令可以根据实际情况在ActionFixture的子类中进行创建。上述的3个命令是

Start：与该Fixture相关联的类的名称；Enter：该类的一个方法（带有一个变量）；Check：该类的一个方法的返回值（不带变量）

为该表格创建一个ActionFixture类如下：

```
package cstc.fitexam.coffeemaker;
import fit.ActionFixture;
public class AddInventory extends ActionFixture {
    private CoffeeMaker cm = new CoffeeMaker();
    private Inventory i = cm.checkInventory();
    public void unitsCoffee(int coffee) {
      cm.addInventory(coffee,0,0,0);
    }
    public void unitsMilk(int milk) {
      cm.addInventory(0,milk,0,0);
    }
    public void unitsSugar(int sugar) {
      cm.addInventory(0,0,sugar,0);
    }
    public void unitsChocolate(int chocolate) {
      cm.addInventory(0,0,0,chocolate);
    }
    public int coffeeInventory() {
      return i.getCoffee();
    }
    public int milkInventory() {
      return i.getMilk();
    }
    public int sugarInventory() {
      return i.getSugar();
    }
    public int chocolateInventory() {
      return i.getChocolate();
    }
}
```

此 fixture 将调用下面清单中显示的 SUT（被测对象） CoffeeMaker 类，然后调用该类上的相关方法。

```
package cstc.fitexam.ffeemaker;
    public class CoffeeMaker {
    /**
    * Array of recipes in coffee maker
    */
    private Recipe [] recipeArray;
    /** Number of recipes in coffee maker */
    private final int NUM_RECIPES = 4;
    /** Array describing if the array is full */
    private boolean [] recipeFull;
    /** Inventory of the coffee maker */
    private Inventory inventory;
    /**
    * Constructor for the coffee maker
    *
    */
    public CoffeeMaker() {
    recipeArray = new Recipe[NUM_RECIPES];
    recipeFull = new boolean[NUM_RECIPES];
    for(int i = 0; i < NUM_RECIPES; i++) {
    recipeArray[i] = new Recipe();
    recipeFull[i] = false;
    }
    inventory = new Inventory();
    }
    ...
```

运行结果如表 3.2 所示：

表 3.2 FIT 的执行脚本

```
fit.ActionFixture
    start Cstc.fitexam.coffeemaker.AddInventory
    enter units coffee 3
    enter units milk 5
    enter units sugar 6
    enter units chocolate 7
    check coffee inventory 18
    check milk inventory 20
    check sugar inventory 21 expected
    _____15 actual
    check chocolate inventory 22
```

在表 3.2 中，第 3 列的结果“21 expected” 表明预期的结果应该是 21，而实

际结果是15。FIT给予了客户和程序员一个关于软件的精确交流的方法。客户所给的具体的例子让程序员能深刻理解将要构建的产品。程序员的对于装置的工作和软件可以让客户给出不同的例子进行试验来获取对于软件如何真正工作更深入的了解。这样通过一起工作，整个团队可以学会更多关于产品的内容并产生更好的结果。

3.6 混合型测试框架

最普遍的使用的测试框架并不是上述的任何单一框架，而是结合上述技术的一种组合式框架，抽取各个的长处来弥补组合引入的不足。这种复合的测试框架是多数单一技术在实际项目中进化得来的。图3.4展示了一个可能的方法，将不同的技术整合起来。

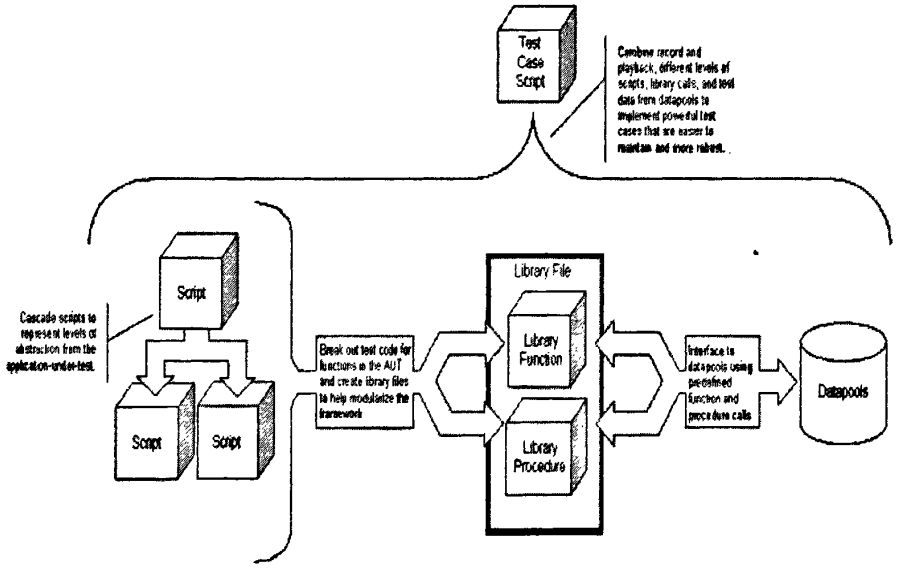

图 3.4 组合型自动化测试框架

3.7 本章小结

本章首先阐述了五种不同的自动化测试框架，每种框架包含的方法都是实际测试过程中有可用来替代仅依靠捕捉回放实现自动化的工具。无论使用一种或多种的组合，可以是将SQABasic库的脚本文件组合进模块化的框架，还是使用数据池来实现数据驱动技术，抑或扩展Rational工具来和其他数据源做整合，关键在于选择最适合的框架，最好的方法是实际去体验各种方法的不同。因为自动化测试框架在某种程度上更是一种实践的艺术，只有通过实践才能体会到其中的

不同和奥妙。

本章接下来介绍了采用关键字驱动的两个开源框架：Selenium和FitNess。两者优秀的设计思想给自动化测试带来了新的方法。最后简略介绍了组合型框架模型。下一章节将对自动化测试框架做进一步的比较说明。

第4章 自动化测试框架的比较

本章对三种主要的设计框架（模块化，数据驱动，关键字驱动）的优点和不足进一步做深入分析。

4.1 模块化框架

模块化设计看似简单，却是最为行之有效的手段之一，因此在测试脚本设计中也是应用最频繁的一个设计模式。虽然模式使用简单，但是如果用的不得当也会造成代码的复杂化，或者过多的冗余，就好像是一把双刃剑。下面将要涉及到的就是笔者认为最主要的三个设计原则，在编写测试脚本时，只要时时遵照这三条原则就不容易引入大的问题。

第一个模块化框架的原则是一次都只关注在一个逻辑功能模块上，或者只抽取一个模块。使用抽取功能，可以尝试减少并且剔除没有直接与功能关联的代码细节。在模块化中，需要关心的是剔除测试用例中的控制器。所谓控制器就是管理着测试程序的测试用例流程和功能性之间的关系的关联组件。与数据驱动的框架相比，需要更多的关注从测试用例中提取的抽象数据。控制器抽象简单的说就是动作的抽象。

第二个原则与抽象有着十分紧密的联系。这个抽象就是封装原则。在封装中，关注的是包装程序的元素（脚本，类等等）放入一个更大更抽象的实体（其他脚本和 类）。封装会组织起相关的任务和操作，同时也会保护脚本不受到代码变更的影响，通过一个固定的接口可以保护实际实现的动作和调用脚本的方式。这类信息隐藏机制可以保护测试脚本；将所有的脚本按照一定的层次分类集合，因此需要改变测试脚本的时候，相级联的脚本都能不受影响。

第三条原则是分离关注点。所谓分离关注点就是将程序分解到一个个独立功能的代码块，他们之间的功能重复尽量是最小的。这个原则强调的是实现封装。典型的，使用分离概念原则，计划如何分割和组合被测程序的功能和行为的测试代码。

上述三个原则也使得模块化是最容易掌握和理解的。三个原则表述都是有名的编码策略，许多程序员甚至在使用的时候都不用刻意的去记忆。模块化的目标

是使用这些原则来增强测试套件的可维护性和可扩展性。

4.2 模块化的优点和不足

这三个类型的框架中，模块化是最简单和最容易实现的。同时他也最容易和其他框架协作，这就是为什么总是将模块化和数据驱动放在一起举例。总的说来，实现模块化只需要少量的技术能力储备。只要能理解概念就能方便的在测试脚本中使用。

模块化的主要优点是重用和减少重用带来的维护成本。和录制回放功能相比较，模块化应该能产出更具可读性的脚本，可以容易的被别人读懂或调试。与录制回放相比，模块化通常要花费更多的成本，但是带来的好处是维护的成本要低很多。这就意味着需要去理解测试脚本会使用多久以及完全理解执行模块化所引入的成本。成本收益分析会很好的帮助理解是否要在当前项目中使用模块化。机会是，即使整个框架不被模块化，还是会将核心功能分离出来放到辅助类里。

脚本模块化同样也有一些问题。模块化的结果是引入了依赖性，结果是如果被测程序有阻碍性的 bug，就会使整个测试序列失效。所谓阻碍性的 bug，是只有当这个问题被修复，其他功能才能被继续执行的缺陷或问题。脚本依赖将会导致脚本瓶颈的发生，一旦界面改变了，脚本就会失效，但是如果大多数的脚本都调用了这个模块，怎么办？例如，如果用来处理购物车的脚本由于开发人员改动了商品在其中的显示方式而变得不可用了，为什么检查订单状态的测试用例要关注这点变动呢？不能是它仅仅点击‘下订单’按钮就能看到结果，而不需要验证表格的格式么？这并不是这个脚本希望测试的部分。使用模块化框架，如果第一个脚本失败了，大多数跟着调用他的脚本都可能会失败。

使用模块化，应用程序的状态和数据共享可能变成一个问题。如果一个模块失败了，对依赖于他的模块来说意味着什么？如何可以检查一个订单的状态，但是这个订单还没有生成，如果不用搜索功能，怎么将商品加入到购物车？共享数据和状态的复杂性可能引起调试脚本的复杂性，为了搞清楚到底是脚本的问题还是被测程序的问题总是很费时间。这些问题可能引起可靠性问题。

4.3 数据驱动测试框架

数据的选择是数据驱动测试中的第一步。选择驱动被测程序执行所需的执行步骤，或者表示输入到系统中的测试数据序列，或者两者兼有。这个段落将简要的

分析五个不同的方式来选择测试数据：基于风险，基于需求，基于可用性，使用产品发布后的数据，或者使用随机生成数据。

基于风险选择数据：测试数据选择的首要标准是风险。当指认一个风险是，考虑什么会出错。始终寻找在可控的时间和预算下可能发生的导致减少提交正确的功能以及高质量的项目代码的事件。有三种方法来分类风险：

- 风险带来的负面影响--与计划的偏差，实际成效，或者项目计划的花费（如果风险可以被成本化）
- 风险发生的可能性--风险发生的可能性（通常表示成百分比）
- 风险暴露率--负面影响与发生可能性的乘积

基于需求选择数据：可以选择那些直接针对需求而设计的测试数据。寻找应用程序中可以用来测试新功能、容错能力以及安全特性的数据，考虑下面2个问题：

1 如果应用程序有不同的角色，用什么数据来验证每一种角色？

2 哪种特性希望包括进测试覆盖范围，以及你需要使用什么数据？

另外，考虑被测程序在各种不同部署目标机器、环境中会有什么样的影响。问题的校验列表应该要包含被测应用程序和目标环境。哪些数据是需要去测试的：

- 硬件设备
- 设备驱动
- 操作系统
- 网络和通信软件
- 第三方软件组（例如email软件，因特网浏览器）
- 各种配置和设置相关的组建以及各种组合
- 全球化

基于可用性选择数据：可能要使用已经存在的数据，这些包括：

- 生产环境中的数据（在下一部分中深入讨论），如果是容易获取的格式封装起来的
- 从之前版本中获得的数据
- 项目中手工测试的表格数据（文件上传）
- 公司中其他类似项目中得来的数据

- 从其他资源获得的数据（在最后一部分讨论）

主要的思路是，如果数据是容易获取的，并且有意义并且容易使用，包括这些数据在测试中可以节省时间和金钱。笔者特别强调可用性和有意义，因为这一点很重要，要求使得不选择已经有的数据的原因是因为那些是不好的数据。

使用生产环境中的数据：另一个策略是使用的测试数据是实际生产环境的数据。尽管也许不应该仅仅依赖与那些数据，但是这同时也是自动化测试场景中最丰富资源的由来，因为这些数据代表了真实环境中，实际操作得来的数据，同时因为他尽最大可能提供了不用场景的数据。在一个项目中，大约每周读取一次生产数据，同时几乎没有花费任何额外精力获取了 300 到 500 个测试场景。可以装载这些数据直接应用到测试环境中，将它们读到数据文件中用作后来的处理，就能立即了解是否测试系统与生产环境中的系统运行的一样好。这个技术特别适用于发现浮点数值问题，转换率问题，以及数据类型允许的长度问题。但是也许公司有种种制度限制使用实际数据用于测试环境，并且生产环境中的数据难以包括一些用来测试特殊的用例。特别是，如果是外包某些测试工作，必须要仔细参考公司对测试数据的规范，因为可能会涉及到法律上的一些问题。即使不能直接拿生产环境的数据来用，但是通过加工处理，还是可以得到一些加工过的数据，并且所做的修改对系统测试几乎没有任何影响。

使用随机数据生成的数据：很多工具包括了测试数据生成器。随机数据生成器可以用来产生大量的定制数据（依赖于你测试的是什么以及你打算花多长时间来生成）。例如，如果需要测试金融信息中的进位错误，可以：

- 随机生成大量的数据
- 将数据实际输入应用程序
- 通过 windows 计算器运行同样的运算（假设 windows 计算器没有进位错误）
- 比较结果，确保他们匹配

如果需要，用工具生成 500 组不同的用户姓名，地址等信息（TestManager，Excel，GoogleSets 等等）应该要避免尝试一次性为测试用例选择好数据。相反，使用增量和循环的方法来处理测试数据，关注于当前测试周期中认为最有可能产生有用的评估信息。这能帮助避免将太多时间投入到单一的数据集或者同种类型的测

试数据的风险，将花在测试方法上的精力最小化。忽略那些无关部分的测试数据，对系统测试的影响将被证明是微不足道的。

4.4 数据驱动的优点和不足

这里提到的三种测试框架（模块化，数据驱动，关键字驱动），对非技术人员来讲，数据驱动框架是最容易实现的，原因是有许多类似与 Rational Functional Tester 的工具支持这种框架。另外一个原因是，这类框架非常强大。使用半自动化生成的文档，使用了更少的代码创建了大量丰富的测试用例，以及允许做各种组合测试，否则需要花费大量时间来做累赘的手工覆盖测试。

数据驱动框架的主要优点是减少了创建大量不同类型应用程序测试用例的成本。在金融服务或者保险行业应用程序中，数据驱动的利用价值非常高。许多时候用作金融软件或者保险业的测试数据都是小的繁琐的。测试人员需要关注最后的结果是经过了上千组数据复杂计算后的结果。但是对于那些数据和特定的用例依赖性非常重，数据和界面关系很紧密的项目就不适合用数据驱动来做。

对数据驱动来说，最主要的缺点是它很容易创建新的用例。有时候这会使得测试用例难以管理，因为容易创建，使得测试用例在数量上迅速膨胀。这可能并不是太坏的一件事，对自动化来说，越多的测试数据对测试完整性来说越是完善。需要注意的是在维护脚本的时候，不要因为测试数据多而使得脚本的维护成本大大增加。对新创建的用例来说，数据非常清晰也非常容易说明问题，但是随着一次又一次对脚本的维护，需要确定维护的成本不至于太高，也不会对测试脚本的有效性影响太大。

4.5 关键字驱动测试框架

关键字驱动测试（有时候称作表格驱动测试）指的是一种由数据表格和关键字驱动的自动化测试框架。它和数据驱动测试有些不同（但是由于经常和数据驱动测试结合起来实现，使得两者的分别有些模糊）。关键字驱动关注在动作而不是数据。在关键字驱动中，测试流程由表格里的预先定义好的关键字数据驱动。关键字驱动脚本和手动测试脚本不论在结构还是文字内容上看上去很相似，由于关键字驱动方式是笔者着手设计的框架的主要思想来源，因此下文将着重介绍这一思想的独特和强大之处。

继续在第三章提到的 Windows 计算器作为例子，这一次用关键字驱动模型来

实现。

表 4.1 Windows 计算器的关键字表格

Window	Control	Action	Argument
Calculator	Menu		
Calculator	Pushbutton	Click	1
Calculator	Pushbutton	Click	+
Calculator	Pushbutton	Click	3
Calculator	Pushbutton	Click	=
Calculator		Verify result	4
Calculator		Clear	
Calculator	Pushbutton	Click	6
Calculator	Pushbutton	Click	-
Calculator	Pushbutton	Click	3
Calculator	Pushbutton	Click	=
Calculator		Verify result	3

表格的四列分别代表：window：正在操作的应用程序；control：在窗口中处理的对象；action：在那个对象上做的操作；argument：执行某个操作所需要的参数在这个例子中，Window 列包含应用程序窗口的名字（在本例中都是在一个计算器的界面中）。Control 列列出了鼠标所操作的控件名。Action 列列出了鼠标做出的操作。最后一列，参数列，指出了操作控件所带的特殊参数（1，2，3，5，+，-，等等）。这个表展示了一个完整的测试用例；当然可以用更复杂的表格来演示一个完整的测试，在下一章中，将会看到用表格驱动的专家—Fit 框架，实现的更为复杂的表格组合。

有了类似表 4.1 的列表，就可以着手写一个驱动程序来分析并执行这个列表所列的动作。表 4.2 展示了一些 Windows 计算器测试的伪代码。

表 4.2 Windows 计算机测试代码（伪代码）

```
Main Script / Program
{
Connect to data tables.
Read in row and parse out values.
Pass values to appropriate module.
Close connection to data tables.
```

```
}
Menu Module
{
Set focus to window.
Select the menu pad option.
Return.
}
Pushbutton Module
{
Set focus to window.
Push the button based on argument.
Return.
}
Verify Result Module
{
Set focus to window.
Get contents from label.
Compare contents with argument value.
Log results.
Return.
}
```

从这个例子中，可以看到这个框架只用很少的代码就能生成大量的测试用例。数据表格只是用来产生单独的测试用例，驱动程序的代码可以重复使用。

关键字驱动的一大卖点是使得非自动化测试人员可以参与写自动化测试脚本。参看前面这个例子就能发现，不用去考虑如何抓取和分析脚本数据的代码，也可以想象出如何对 Windows 计算器做测试。

就像 Danny Faught 在他的“关键字驱动测试”[16]一文中指出的那样，这个层次的抽象分离了驱动接口和应用程序接口。“虽然范例脚本使用了基于图形化用户界面的接口，但是用户接口可以是 API，web 服务或者是任何其他接口。”

Danny指出在选择关键字的时候最高要避免任何对用户接口的假设。从而自动化测试人员可以不改动关键字脚本的前提下更改接口驱动。传统上，手动测试人员或者业务分析人员撰写关键字表格，自动化测试人员编写驱动代码。从驱动代码的角度来看，过程和编写数据驱动的测试是类似的。目前我们做的是只需要用条件判断和组合判 断在驱动脚本数据中搜索关键字值。

4.6 关键字驱动框架的优点和不足

在上边提到的所有自动化框架中，关键字驱动是最复杂最难以实现的一种。因为它包含了很多循环，条件判断，类组合，辅助脚本等等。但是另一方面，关键字驱动框架非常强大，对于非技术人员或者非测试人员，尤其是业务分析员来说，关键字驱动框架提供了一种非常容易的创建测试用例的方式。取代了其他框架中用大量代码来实现不同的测试用例，关键字驱动的方式用简单的文档取代了复杂的代码，使得测试用例更加易于组织和管理[17]

Danny 在他的文章中准确的总结了这种框架主要的优缺点："如果你不是一个编程人员，但是你的公司使用了关键字驱动的测试框架，你仍然能够实现自动化测试脚本。我和一些使用过关键字驱动的人员的谈话中了解到，让 非技术人员撰写关键字驱动的脚本并不是非常实际的一件事，但是也有其他人说这很容易实现。关于这点的争论一直存在。管理人员需要理解，他们仍然需要程序员 来实现和维护这个框架，这部分的开销必不可少。正真的好处是，这个框架能够提高可维护性。关键字框架使用的是长期实践中形成的模块化自动化测试方式，所以 界面的改动对它的影响是最轻的。仅仅这个优势就足够说明关键字驱动测试是有效的，即使所有的测试设计这都是专家级的程序员。"

4.7 本章小结

本章讨论了自动化测试框架的不同设计思想之间的关联和各自的优缺点。首先是最容易理解的模块化脚本技术，将可以重用的脚本以文件或者类的形式封装起来，供其他脚本调用。随后介绍了数据驱动技术，将测试流程与测试数据解耦合，获得了灵活性，同样的一个流程可能对应上千组数据，有了数据驱动的架构使得重复测试海量数据成为现实。同时，针对如何收集数据，将什么数据放入被测对象集合中也做了具体分析。接下来介绍了基于关键字的的测试手段，将测试动作与测试脚本分离，使得自动化测试设计人员与用例撰写人员独立，即使不懂

自动化技术的业务人员也能参与到自动化测试脚本的设计中来。任何实际的测试脚本都是融合了不同技术的组合体，对应不同的技术难点，不同的框架各有不同的应对手段，没有绝对最好的解决方案，唯有最适合项目的手段和方法。

第5章 自动化测试框架工具设计与实现

5.1 测试框架背景介绍

本章重点介绍笔者在实际项目中使用的自动化测试框架的设计和实现技术细节。笔者所参与的项目是某银行的基金交易系统，该系统主要由三个模块组成：管理模块，投资者模块，基金发行者模块。其中的管理模块主要功能是管理各发行机构以及其管理的基金、投资者账号和投资者权限、投资管理账号等若干应用；投资者模块主要针对机构投资者购买、赎回基金，管理投资帐户，查看投资报表等功能而开发；基金发行者模块主要管理基金交易中的各个状态变化，监管各个账号的交易情况，查看各种数据报表。总体来说，该项目是一个典型的 J2EE 项目，目前的项目是从早先的银行交易系统移植到新的技术平台上。因此，很多的业务都能通过上传和下载文件来完成的，保证和原来系统的向下兼容。在实际测试过程中，由于上传的数据繁多，组合情况复杂，非常不适合手工测试，并且因为操作流程较为单一，很适合用自动化技术来实现上传下载的回归测试。在框架技术的选择上，笔者考虑到上传功能的用例相对简练，但是测试数据相当庞大，采用了关键字驱动的框架，结合数据驱动将上传数据整合到数据池中，并采用模块化，解决了需要在开发、测试、用户体验测试等不同平台上测试相同功能的代码重用问题，并且为今后自动化回归测试和代码持续集成打下了较好的基础，在本文第 6 章第 3 节将就这一问题展开讨论。下文就笔者实际研究中遇到的技术问题和解决方案做一具体的分析。

5.2 底层架构——IBM STAF

为了避免重复发明轮子，笔者在实现自己的框架时复用了 STAF 的设计架构--发端于 IBM 的自动化测试框架，该架构提供了灵活的集成测试功能，并且和笔者所使用的自动化测试工具 Rational RFT 能无缝的衔接。具体来讲，在 RFT 上无论用关键字驱动还是数据驱动实现的代码，都能够在 STAF 框架中重复自动化调用，所有产生的缺陷记录、测试结果都能由 STAF 取出来存储到指定的目录中，并且用邮件等方式发送到开发和 QA 的桌面系统中。

STAF 是开源、跨平台、支持多语言的自动化测试框架，它围绕于组件重用的

理念，通过服务调用（比如处理调用、资源管理、登陆、监视等）帮助开发人员省去繁琐的自动化架构建设工作，自动化开发人员只需集中精力在自身自动化实施上。STAF为自动化测试建立了基础，在高层解决方案提供一种可插拔的机制，支持多种平台与多种语言。[18]

使用STAF可快速构造自动化测试环境，STAF的服务调用系统也让自动化测试人员创建自动用例与管理自动用例更加方便。STAF在功能级别实施服务调用，各个服务端点（称作STAF客户端）是对等的，从一个端点可直接调用另一端点（在另一台机器运行的程序）提供的服务。因此用STAF来实现持续集成是非常有效和出色的，由于日构建思想的兴起，项目组计划在每天晚上进行每日构建，所有代码编译的结果信息都会记录，并且以电子邮件的方式发送给项目组的每个人。如果项目能顺利通过编译，并且自动发布到服务器上，接下来就是执行自动化回归测试。

5.3 测试驱动整体架构

根据软件测试理论，自动化测试用例在软件开发的需求说明完成后就可以书写。当前流行的测试框架，采用的脚本技术是数据驱动脚本技术中的关键字驱动脚本。数据驱动脚本技术将测试输入存储在独立的（数据）文件中，而不是存储在脚本中，脚本中只存放控制信息。执行测试时，从文件中而不是直接从脚本中读取测试输入。这种方法的最大好处是同一个脚本可以运行不同的测试。关键字驱动脚本是较复杂的数据驱动技术的逻辑扩展。数据驱动技术的限制是每个测试用例执行的导航和操作必须一样，测试的逻辑“知识”建立在数据文件和控制脚本中，因此两者需要同步。而关键字驱动技术可以将数据文件变成测试用例的描述，用一系列关键字指定要执行的任务。控制脚本可以解释关键字，但这是在控制脚本之外完成的，所以要求一个附加的技术实现层。[19]

关键字驱动方法的优点是显而易见的。通常这种方法所需脚本数量是随软件的规模而不是测试的数量而变化的。因此，可以不用增加脚本的数量实现很多的测试，只需要替换基本的应用支持脚本。这样大大减少了脚本维护开销，不仅加速了自动测试的实现，而且可以由不会编程的测试人员来完成测试。使用几百个脚本就可以完成上千个测试用例。根据以上的技术实现要求，该模型将测试用例的数据文件书写到Excel 表中，并设置关键字，通过编写测试脚本匹配关键字生

成自动测试所需要的数据文件以及控制信息文件，从而达到自动化测试的目的。

5.3.1 自动化软件测试模型的架构分析

正如前文所分析的，一个好的自动化测试框架离不开下面这些要素：

1）界面元素名与测试内部对象名的分离

我们可以在被测应用程序和录制生成的测试脚本之间增加一个抽象层，它可以将界面上的所有元素映射成相对应的一个逻辑对象，测试针对这些逻辑对象进行，界面元素的改变只会影响映射表，而不会影响测试。

2）测试描述与具体实现细节的分离

我们应该把测试描述和测试的具体实现细节分离开来。测试描述只说明软件测试要做什么以及期待什么样的结果，而不管怎样执行测试或怎样证实结果。这样做是因为测试的实现细节通常和特定的平台，以及特定的测试执行工具有着密切的联系。这种分离使得测试描述对于应用实现细节是不敏感的，而且有利于测试在工具和平台间的移植。

3）脚本与数据的分离

最后，可以把测试执行过程中所需的测试数据从脚本中提取出来，在运行时测试脚本再从数据存放处读取预先定制好的数据，这样脚本和数据可以独立维护。

图 5.1 描述了这个模型的结构：

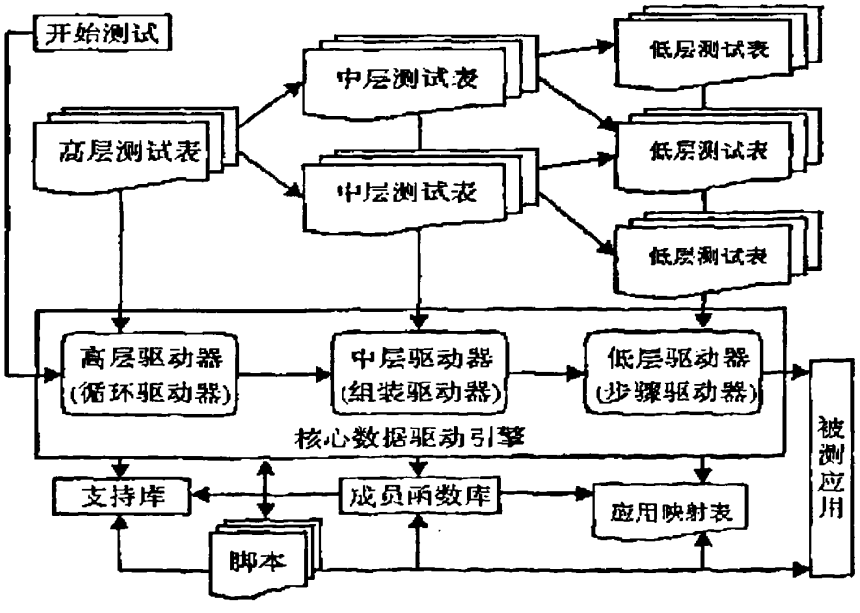

图 5.1 基于关键字的自动化测试模型架构图

这个模型由测试表、核心数据驱动引擎、成员函数库、支持库、应用映射表组成，分别介绍如下：

1. 测试表（Test Tables）：保存测试数据和关键字，分为高层测试表、中层测试表、低层测试表。其中，下层的测试表被上层的测试表所调用。
2. 核心数据驱动引擎（Core Data Driven Engine）：与测试表对应，分为高层驱动器（也叫循环驱动器）、中层驱动器（也叫组装驱动器）和低层驱动器（也叫步骤驱动器）。上层的驱动器读取相应测试表的关键字逐级传递给下层的驱动器，最后由低层驱动器调用关键字库中的指令对应的组件函数来执行。
3. 组件函数库（Component Function）：组件函数实现了用户对界面对象的各种操作指令，它在被测应用和自动化工具之间提供了一个隔离层。
4. 支持库（Support Libraries）：通用的程序和工具库，提供诸如数据库访问、字符串操作、文件访问、日志记录等基础性的支持功能。
5. 应用映射表（Application Map）：对应用中的对象定义一套命名规范，将这些实际对象的名字和自动化工具识别的对象名联系起来，形成映射表，从而使应用对象元素和测试对象名分离，提高了脚本的可维护性。

5.4 测试用例的搭建

笔者所设计的自动化框架雏形也正是依据以上这个模型进行创建的。在有计划的实现文件上传功能自动化测试的同时，将好的设计模式应用到框架设计中，甚至是看似和自动化工具毫无关联的测试用例撰写。通常一份好的测试用例是自动化测试顺利进行的基础，虽然不能直接与自动化脚本打交道，但是对后续工作的开展绝对是一个重要的环节。根据需求用例提取，并条理化细分得到的用例通常还包含一份用例的组合测试流程（Test Scenario），也就是高层测试用例，它的存在使得测试的功能点更明确，测试流程更清晰，有利于自动化脚本的编写。难以想象，一个自动化开发人员不了解测试的步骤过程，就能动手开发自动化测试脚本。当拥有了不同的测试流程，就可以用来组合成更复杂的测试场景，来验证同样很复杂的业务逻辑。需要注意的一点是，由于在测试场景中包含了大量重复的操作，这些操作常常可以被移动到单独的测试脚本库，从而在不同测试流程中来调用库方法，即实现了脚本的模块化，又便于后期的维护。

在建立关键字驱动框架中，笔者遇到的第一个问题就是如何将测试用例转换成机器可以识别的关键字脚本。为了说明这个问题的解决过程，先来看下图 5.1 所示的文件上传系统测试用例结构。

8. UC INV-6 Upload Orders via GUI

8.1 Preconditions

PREC1: The Fund Investor has logged into Fund Connect with a valid username and password

8.2 Basic Flow of Events

BF1. The Investor clicks on the Upload Orders link from the Investor Home page.

BF2. Fund Connect displays the Upload Multiple Orders page.

BF3. The Investor selects the Browse button, which displays a file dialog box. The Investor selects a file and then selects the Submit button from the Upload Multiple Orders page.

BF4. Fund Connect will read the file and parse the contents. As each order is read in, Fund Connect will validate the trade and perform cutoff time validation. Once all orders have been processed, Fund Connect will display a summary of the upload in the Multiple Orders Upload Summary screen.

8.3 Post Conditions

8.4 Alternative Flows

AF1. Upload Invalid File

Entry Condition: BF3, the Investor enters an invalid filename and selects the Submit button from the Upload Multiple Orders page.

AF1.1 Fund Connect will display a file uploaded error to the Investor.

8.5 Key Scenarios

KS1. Upload Orders via the GUI	BF
KS2. Upload Invalid File	BF1-BF3 -> AF1

图 5.2 用户上传文件的测试流程图

可以看到改测试流程由一个进入条件（PREC1）和四个基本事件流（BF）组成了第一个关键流程（KS1）,同时由三个基本事件流和一个异常事件（AF）组成的第二个关键流程（KS2）。这个测试流程片段作为一份高层测试表覆盖了应用程序基本的文件上传功能测试点，由这个高层测试表，可以获得相应的中层测试表，如图 5.2。

1.1.1 Upload "Orders" File

Steps:

1. Admin user navigates to the INV module.
2. Admin user clicks the Home tab, and navigates to the "Investment Options" page.
3. Admin user clicks the "upload orders" button then clicks the "Browse..." button.
4. Admin user selects a csv file which includes the testing orders data.
5. Admin user clicks "Submit" button to upload the file.
6. Fund Connect displays the summary screen which will display the uploading information message

Sample test data

Precondition: INV user has EXEC or EXEC/Approval permission over the fund ABNfund01/02/03

Case #	FP_Name	Fund_Name	Inv_Account	Side	Qty_Type	Quantity	Value_Date
1.	FP_BIN	testfund	ABN Account01	BUY	Currency	100	8/22/2006
2.	ABN AMRO	ABNfund01	ABN Account01	BUY	Shares	30	8/22/2006
3.	ABN AMRO	ABNfund02	ABN Account01	BUY	Currency	133.33	8/22/2006
4.	ABN AMRO	ABNfund03	ABN Account01	BUY	Shares	22.22	8/22/2006
5.	ABN AMRO	ABNfund03	ABN Account01	buy	Shares	22.22	8/22/2006
6.	ABN AMRO	ABNfund03	ABN Account01	buy	SHARES	22.22	8/22/2006
7.		ABNfund03	ABN Account01	BUY	Shares	22.22	8/22/2006
8.	ABN AMRO		ABN Account01	BUY	Shares	22.22	8/22/2006
9.	ABN AMRO	ABNfund03		BUY	Shares	22.22	8/22/2006
10.	ABN AMRO	ABNfund03	ABN Account01		Shares	22.22	8/22/2006

图 5.3 中层测试用例图

这份测试用例简单来看分为两部分，操作步骤和测试数据。其中操作步骤部分详细的列出了实现了高层用例中基本流程的操作步骤，在下文就将具体介绍如何将这部分自动转化成关键字驱动表。测试数据部分列出的是为了验证系统要求的功能所设计的各种用例数据，这部分的数据最终将会被导入到自动化脚本中的数据池中，见图 5.3.

图 5.4 用于上传文件的数据池

需要说明的是数据池里的每一行代表一种测试用例，同时针对每一种用例列出了正确的测试结果，其中的一些特别设计的标签是用于替换类似日期时间、随机数值、空白和用例总数等和应用程序关联的数据。在实际应用中，笔者发现固定的测试数据远远不能满足这类需要大量测试数据的用例，即使是使用一些数据模拟

库，也不能适应类似于时间日期一类灵活变化的数据，因此，在数据池中需要用一些标签来满足这些灵活的变化。这么做带来的好处是灵活性和可扩展性，坏处是代码的维护成本增加。因此是否要在自动化中使用这一技术还是要根据项目具体的情况来权衡。

在上文中，高层测试用例和中层用例都是手工完成的，一般来说是测试工程师根据需求用例来撰写的，一份好的测试用例能详细的记录用户执行的操作，应用程序界面的变化和期望的结果。接下来，笔者将介绍如何从一份详细的用例中自动生成关键字驱动脚本，主要用到的技术是关键字语法分析和界面控件的匹配管理。

如果是一份详细的用例，要从中提取关键字并不是十分困难，因为一般来说一份完整的用例大概会包含20-30个常用的动词，剩下的名字往往是用户（通常在主语的部分），WEB界面（通常跟在Navigate，Display等动词后面），页面控件（通常跟在Click，Select等动词后面）和测试期望的结果（通常在Verify，Expect等动词后面）。有了这些模式，就能够通过计算机自动的对用例做语法分析。通过对图5.2中的用例进行语法分析，可以对应得到图5.4所示的关键字表。对应的操作界面如图 5.5-5.8 所示，用红框标出来的是和关键字表中 Control 字段对应的页面控件。需要说明的是和用户界面有关的元素的提取，这部分功能的实现和下文要介绍的图形界面—测试元素匹配表相关，简单来说界面上的每个控件和它所在的页面绑定，用某种方式预先将控件从页面上抓下来（可以是项目的某个原型），然后用类似Map的数据结构将界面元素和脚本中的参数域关联起来。

Sequence	Window	Control	Action	Argument
1	Investor Home	Home menu	Click	
2	Investor Home	Upload button	Click	
3	Upload Orders	File InputBox	Click	
4	Upload Orders	File InputBox	Input	<FileName>
5	Upload Orders	Submit button	Click	
6	Summary Page	Result table	Verify result	<Result>
7	Summary Page	Home menu	Click	
10	Investor Home		Verify result	<HomePage>

图 5.5 上传文件的关键字驱动表

图 5.6 步骤1，2对应的页面控件图

图 5.7 步骤3，4，5对应的页面控件图

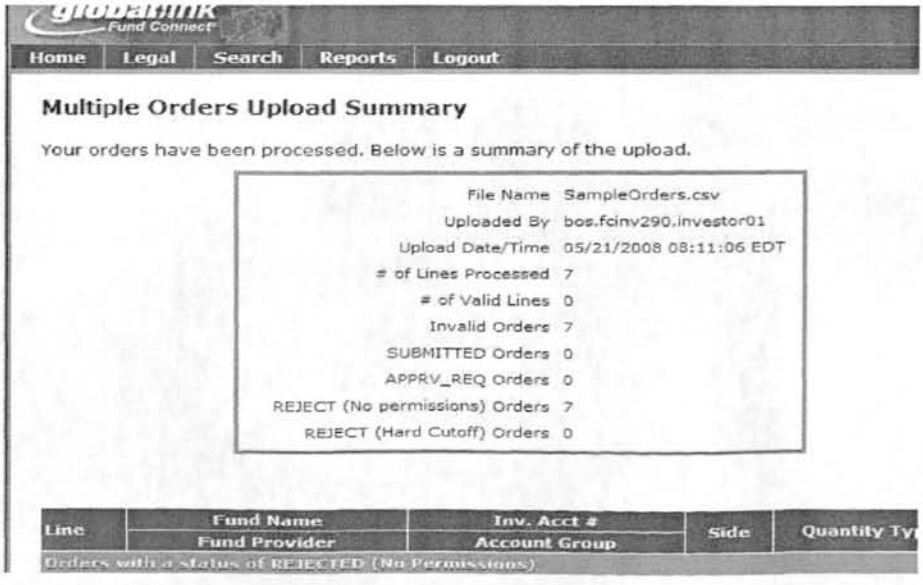

图 5.8 步骤6，7对应的结果控件图

上传结果表中的每一行结果都与数据池中每一行的测试数据对应，因此对结果的验证也可以做到数据池中，用自动化的方法来验证结果。图5.8给出了数据池针对结果验证部分的数据，可以看到每个域的值是由脚本运行中根据上传文件中账单的数量动态计算的，并不是一个固定的值（非0值）。

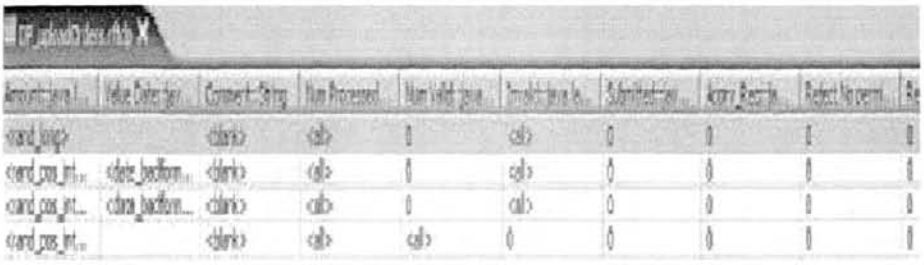

图 5.9 数据池结果验证部分数据

接下来，着重介绍如何将界面元素和关键字控件域上的字段关联起来。在软件开发过程中，用户界面往往是变化最频繁的因素之一，有时候从一个最初的设计原型开始，经过不断的修改，最后产品界面会变得面目全非。笔者并不推荐在业务尚不稳定的情况下开展自动化脚本的编写，但是通过分离界面元素和脚本字段可以有效的应对界面频繁变化，因为通常说来，界面变化主要是CSS格式的变化，控件位置的变化，控件类型的变化，和控件样式的变化。在应用程序需求没有大变动的前提下，如果采用相对灵活的控件管理机制，只需要改动很少一部分脚本，就能让测试程序在不同版本的界面上运行。这里笔者主要提两种管理机制，首先是大而全的管理，来源是Rational Functional Tester的专利技术。主要思想是将界面上的控件所有相关的信息在脚本录制过程中都记录下来，因此一个简单的控件有20个左右的属性，其中最主要的是父窗口ID，控件ID，控件名称，控件坐标等，如图5.5所示。

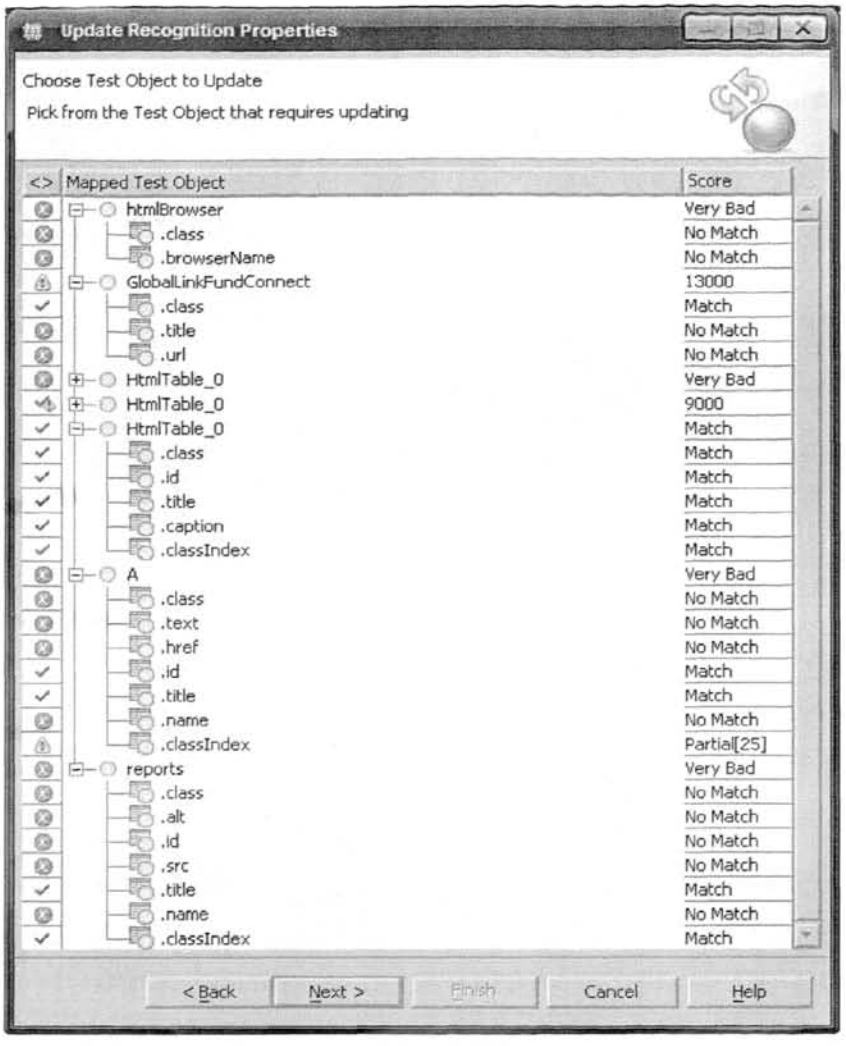

图 5.10 RFT 中记录控件属性的结构图

Rational 针对每一种属性赋予一个权重，在界面元素变化时，如果脚本找不到同样的控件，则通过加权计算出当前元素与记录元素之间的误差，取误差最小的一个控件，如果这种误差在可以容忍的范围内，那么 Rational 就接受这个控件，并且继续在这个控件上操作，如果这个控件侧误差超出了容忍范围，那么 Rational 就将停止当前的操作，抛出异常并退出脚本的执行。

另一种方式则要轻量级的多，笔者在前文对 Selenium 软件的介绍中已经提到过。针对 Web 页面使用 HTML 语言标签组织起来的，虽然不如 XML 那么规范，但是通过 XPath 和 DCom 扔然可以较好的定位一个控件。在实际开发过程中，界面控件的名称和 ID 相对稳定，只是定位坐标和 classID 等属性相对变化较大。

通过 XPath 可以很好的定位一个控件，并且即使控件名称变了，由于这种方式只监控 1-2 个属性，因此管理和维护的代价也会相对小很多。

暂且不论那种方法更好，针对使用脚本元素和界面元素分开管理的方案是两者的共同点，也是很好的一个设计模式。因此在笔者的模型中同样使用了类似 Rational 的方式管理，但是其中的属性要远远少于 Rational 默认的数量，通常在 3-5 个左右就足够识别一个控件了。

由于被测试系统在产品化前要经过开发，测试，用户测试等步骤，而每一个步骤对应的在不同的运行环境中，最明显的是系统的登陆界面都不太一致。由于测试环境有严格的用户验证和认证保护，其界面如图 5.10 所示，而开发环境则省去了复杂的用户认证机制，关注于功能的实现，其界面如图 5.11 所示，另外用户测试界面会为不同的客户单独开发登陆界面。但是除了登陆界面的差别，应用程序交易的操作界面几乎是完全一样的。

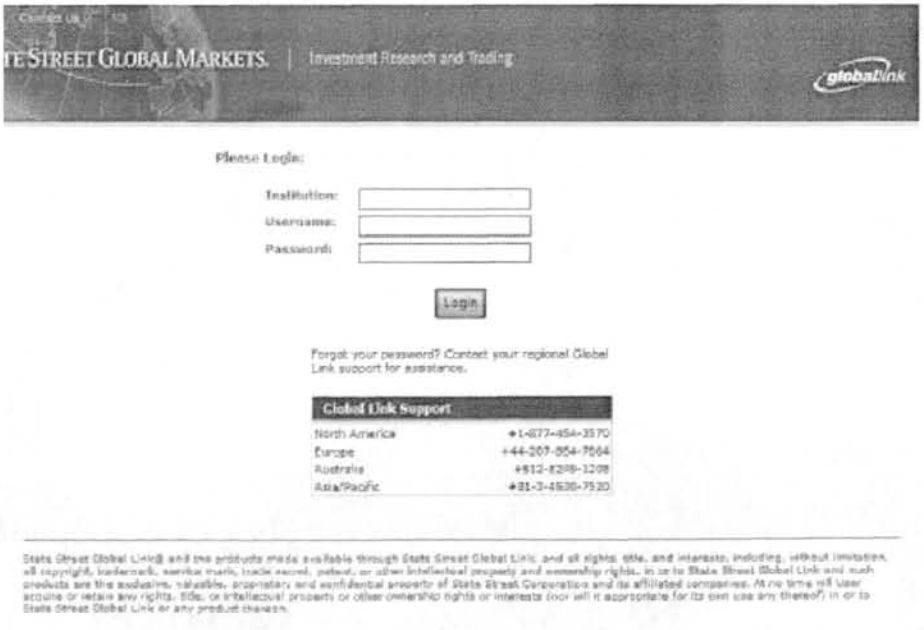

图 5.11 测试环境中的用户登陆界面

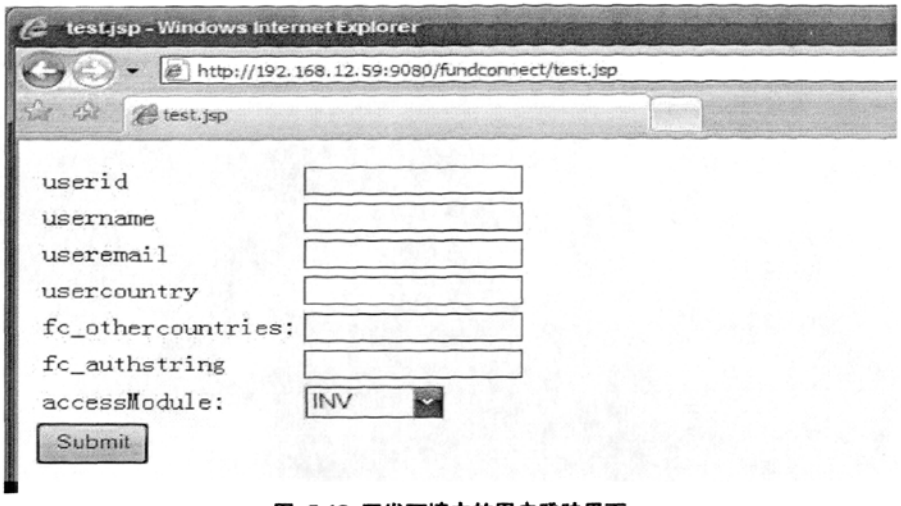

图 5.12 开发环境中的用户登陆界面

那就没有必要为不同的界面维护多份脚本，比较合理的做法是将登陆脚本分离，而业务逻辑验证这块脚本则是大家公用，这就是模块化带来的好处。表 5.1 列出了在业务脚本中调用登录脚本的代码，图 5.12 列出了针对不同环境的登录脚本。

```
//private static final String CSVFILE = "TEMP.CSV";
public void testMain(Object[] args) {
    // callScript("SMLogin.INV_login");

    // Login in for FC 3.0 version only
    callScript("SMLogin.INV_login");

    // prepare the test data file according to download file
    // and test DP;
```

表 5.1 调用登录脚本的业务

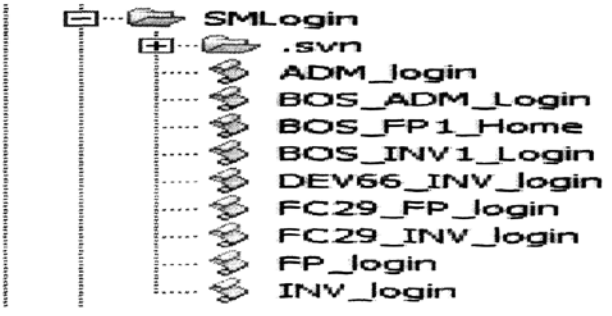

图 5.13 登录脚本

5.5 本章小结

本章主要介绍了笔者自己实现的用关键字驱动的自动化测试框架的设计思想和实现方式。笔者采用了 IBM 提出的 STAF 框架，作为自己设计工具的基础，

达到了高可扩展性和灵活性。同时从前文介绍的大量框架中提取了有利的元素来构建自己的系统，使得该系统在能正确实现文件上传自动化测试功能的基础上，相对获得了更好的性能和可维护性。最后大致介绍了笔者的系统是如何实现文件上传功能的自动化。在下一章中笔者将重点介绍在现有的技术和平台下，如何能最大化自动化技术给业界带来的创造力——智能化平台的构建。

第6章 对当前框架的改进构思——构建智能化平台

6.1 智能化测试平台的构思

软件自动化测试技术日新月异，从早期的手动测试，到后来的录制/回放半自动化测试，到现在的数据驱动、关键字驱动的自动化测试手段。技术的不断创新，弥补了实践中不断发现的缺陷。本章主要针对如何有效的进行自动化回归测试，如何能达到尽可能全的覆盖率，如何管理测试用例和自动化测试脚本，如何更好的组织自动化脚本，尽可能多的重用已有脚本，以及如何设置功能校验点更有效和自动化持续集成回归测试等方面提出新观点，新方法。

6.2 模型驱动的自动化测试

在当前流行的自动化测试中最大的问题是，隐含在手动测试中的随机顺序性被忽略了。一旦应用程序能经受正常的测试用例流程，而不产生明显的错误或者失败时，就可以发布给用户了。当客户拿到产品时，使用了与自动化测试用例不同的流程来操作软件，并且发现了 bug。客户打电话给客服部门，支持部门将错误报告给开发组，然后开发组修复了这个 bug。测试工程师，复查了这个 bug，发现了他们在自动化脚本中漏掉的 case，并且很高兴的将这个缺失给补上了。至少到下一个用户用另一种不同的流程来测试这个功能，并且发现了新的 bug 前，测试脚本都是有效的。但是，问题的发生使得项目经理还是会考虑，是否大量的手动测试并不是一个太坏的办法，至少可以发现一些流程上的问题。

但是，事实上还有另外一种方法，对自动化框架作为一种补充。假设自动化测试知道程序的一个状态和下一个状态间是如何建立联系的。更进一步假设，自动化脚本知 道从当前状态开始，通过一切手段可能达到的下一个状态。通过这个行为模型，自动化脚本可以执行当前的功能，校验正确的结果，随机的选择一条路径执行下一个操作，然后测试下一个状态的功能，再随机继续选择下一个状态。利用路径覆盖的算法，可以达到 100%的覆盖率，弥补了手动测试无法覆盖所有路径的缺陷。这种测试框架被称作基于模型的测试。

基于模型的测试是建立在自动化测试的基础上，模仿随机行为的测试方式。每一个程序的功能（页面）变成了一个“测试状态”，每一个按钮、链接、菜单项

或者命令可以将程序引入下一个界面的就是"变迁行为"。从而，我们的被测应用程序就变成了一个由所有可能的状态，所有可能的变迁行为，以及代表最终结果的目标状态。在 web 应用或者 Java 应用中，一个界面或者消息框都和一个状态关联。对每一个状态的业务逻辑或者应用逻辑的验证和传统的自动化测试没有区别。也就是说，自动化脚本可以被基于模型的框架所容纳，甚至是复用。

最大的区别在于，传统的测试，不管你运行了多少遍，测试的流程总是固定的，没有变化。在模型驱动的测试中，由于测试路径的选择是随机的，计算机对每一种选择做无差别的遍历。对于 Java 或者 Web 应用，有大量的用户访问单一的服务器，这个框架对多用户访问的测试是一个很有价值的工具。

基于模型的测试是对现有自动化测试框架很好的一个扩展。由于某些操作随机化的本性，这个框架覆盖了大量重复普通自动化脚本也不能发现的流程上的一些问题。而且，使用并行化计算的体系结构，同时可以覆盖大量的路径测试。这个模型对拥有大量用户的网站系统的功能测试或者系统测试有很好的覆盖度，避免一些由于常规测试覆盖不全面而导致漏测[20]。

6.3 持续集成中的自动化回归测试

如今软件开发依赖于集体的开发和测试。对于部署和测试人员来说，如何从集中的代码管理工具来获取源代码或者代码的编译包并且自动部署和测试变得非常重要。持续集成能够使开发人员和测试人员在同一个最新的版本上工作。

CruiseControl 是现在流行的持续集成的软件。自动化测试能够大大减轻在持续集成中由于项目代码在不停的构建所引起的测试人员的工作量迅速增加，减少测试过程中人为出现的错误。STAF (STAX) 是一个轻量级的自动化测试框架。提供了一个自动化测试的平台，帮助我们进行自动化测试的更新、编译、部署和测试。STAX 为 STAF 提供了一个执行引擎，帮助加快 STAF 程序的开发和部署。因此利用 STAF 和 STAX 可以减少测试的工作量和复杂度，加快软件测试的流程，缩短测试周期[21]。

笔者结合 CruiseControl 和 STAF (STAX) 来介绍一个复杂环境下的自动化测试方案。使用 CruiseControl 作为自动化测试的入口和调度器，用它来控制 STAX 任务的执行。STAX 通过 STAX 复杂对测试代码进行分发、编译、部署和测试。在整个测试过程中，不仅可以使用 JUnit 测试用例来测试代码，而且可以

使用其他方式的测试用例来测试，比如 Rational Functional Tester 脚本，Robot 脚本等。

用一个典型的基于 web 服务(WebService)的 J2EE 项目为例，WebService 和客户端分别运行于 Windows 和 Linux 平台上的 WebSphere 应用服务器之上。在 WebSphere 应用服务器上需要配置登录认证模块来实现用户访问前的身份认证。

在这一应用场景中，测试团队经过分析，对自动化功能测试提出了以下需求：

- WebService 和客户端需要分别进行功能测试。
- 开发团队提供了用于配置应用程序登录模块（LoginModule）的自动配置部署脚本，该脚本发布到一个 FTP 服务器上。
- WebService 和客户端的源代码以及测试脚本的源代码都存放在 CVS 上，需要在测试执行前进行自动更新。
- 自动化测试执行需要每天自动定时运行。
- 测试执行的结果包括执行日志需要被测试团队和开发团队及时访问到。

6.3.1 自动化测试方案的设计

针对上述需求，可以将这一自动化测试方案分隔为如下图所示的几个逻辑组件：

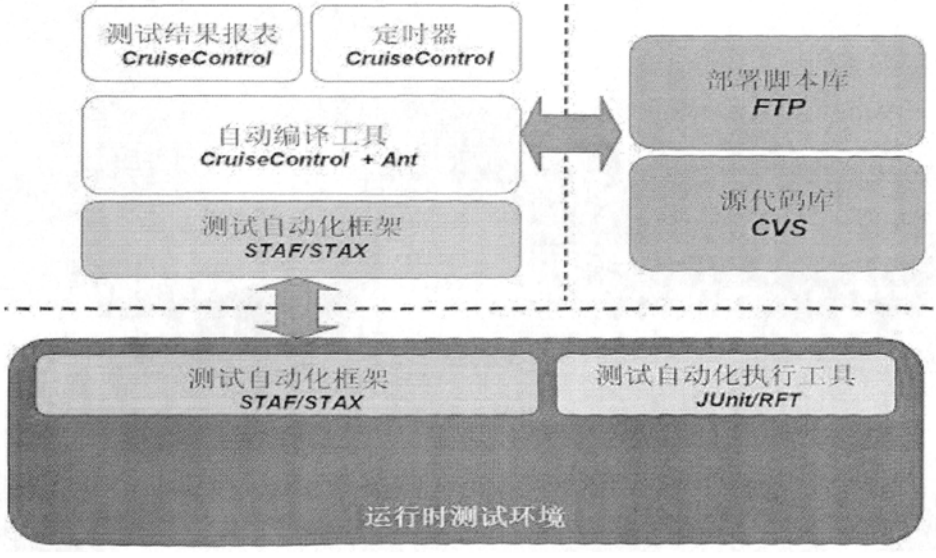

图 6.1 自动化测试方案的逻辑框图

部署脚本库：是指发布配置登录模块的脚本的服务器。在本例中，是一个 FTP 服务器。

源代码库：是指发布被测试应用程序源代码以及测试相关源代码的服务器。在本文中，WebService 和客户端的源代码以及测试脚本的源代码都被发布在一个 CVS 服务器上。

定时器：设置每天特定时间触发，本文中，定时器的功能是 CruiseControl 提供的。

自动编译工具 将从源代码库中获取的测试代码和被测试代码进行编译。本文中，使用 CruiseControl 结合 Ant 作为自动编译工具。

测试自动化框架：负责将编译完成的被测试代码和测试代码部署到测试环境中，并调用测试自动化执行工具来执行测试。在测试执行介绍后，框架还需要收集相应的执行日志，并将执行 结果和日志交由 CruiseControl 进行发布，以供测试和开发人员参考。在本文中，测试框架采用了开源的 STAF、STAX 框架。

测试执行工具 是指进行测试的各种测试工具，本文中，测试工具使用了 JUnit，RFT 等。

测试环境：是指运行被测试代码和测试脚本的环境。在实际应用中测试环境既可能是一个简单的 Windows 服务器，也可能是由许多不同平台，不同类型的服务器组成的复杂环境。本文中，测试环境包括了不同版本不同平台的 WebSphere Application Server，以及运行测试脚本的一台服务器。

测试报表工具：是将测试执行结果和日志进行呈现的工具。由于测试过程完全自动进行，只有将测试结果和日志尽可能清晰的展示在测试和开发人员面前，才能充分发挥自动测试的作用。

从上面对各个逻辑组件的描述中可以看出，测试框架在整个自动化测试方案中位于核心位置。它像控制器一样控制着其他各个组件，同时又像胶水一样把各个组件连结起来协同工作。接下来给出具体的测试方案设计。

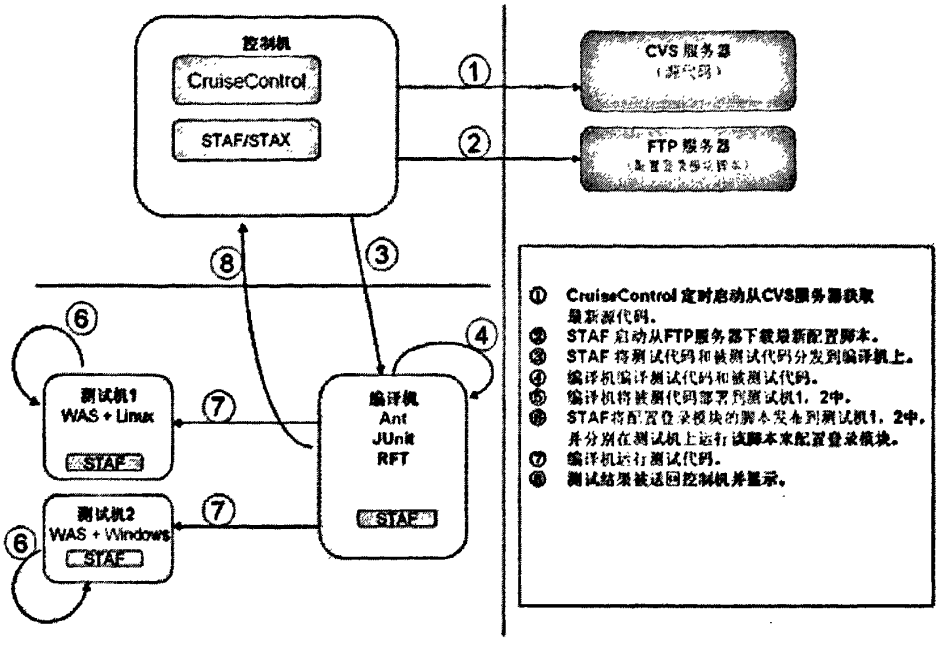

图 6.2 测试方案设计实现

在上图中，

控制机上安装了 CruiseControl 和 STAF、STAX 框架两套工具。

编译机上安装了编译工具 Ant 以及测试执行工具 JUnit 和 RFT。其中 JUnit 测试脚本用来测试 WebService，而 RFT 测试脚本则用来测试和操作客户端。

测试机 1 和测试机 2 则分别用来运行 WebService 和客户端。

该测试方案的具体执行步骤如下：

- CruiseControl 服务器在每日指定的时间自动从 CVS 服务器获取最新测试代码，完成后 CruiseControl 服务器执行一个批处理命令来启动一个 STAF 任务。
- 该 STAF 任务从 FTP 服务器下载最新配置登录模块的脚本到本地目录。
- STAF 将 WebService 和客户端的源代码，测试脚本源代码，以及测试代码分发到编译机上。
- STAF 在编译机上调用编译工具 Ant 来编译被测试代码（编译出包含 WebService 的 war 和包含客户端的 war），测试脚本源代码。
- STAF 将编译出的运行时代码（war）通过 WAS 远程命令部署到测试机 1，2 中。
- STAF 将配置登录模块的脚本发布到测试机 1，2 中，并分别在测试机上运行

该脚本来配置登录模块。

- STAF 在编译机上调用 JUnit 和 RFT 来运行测试代码。
- STAF 将测试结果送回控制机并显示。

6.3.2 使用的产品和技术

STAF/STAX 软件测试自动化框架（Software Testing Automation Framework，简称 STAF）是一个开源的测试自动化框架，它的设计核心理念是称为“服务”的可重用组件（例如，进程调用，资源管理，日志和监控等）。STAX（Software Test Automation eXecution Engine）是基于 STAF 的执行引擎，它采用 XML 格式描述。在 XML 文件中可定义测试工作流，可以实现并行执行、嵌套测试用例、控制运行时间等，STAX 支持 Java 和 Python 模块。STAX 让 STAF 的使用变得更简单，测试人员只需要配置 XML 文件便可以实现 STAF 任务管理。STAX 同时还提供了一个强大的图形界面监控程序，使用该监控程序可以监控并控制正在 STAF 框架中执行的任务。

换一个角度看，STAF 是一种分布式远程调用体系，所具有的以下特色使得它成为了集成测试中非常优秀的框架设计：

1. 将环境需求最小化（包括硬件与软件）
2. 在各种语言中都很容易使用，包括 Java，C/C++，Python，Perl，TCL，及命令行 shell 环境
3. 易于扩展，让用户能方便的创建一个服务插入到 STAF 体系中

STAF 比较适应需要构造复杂测试环境的场合，复杂测试环境通常是分布式的，通过 STAF 将测试任务分发到不同的测试环境去执行，可以方便的测试机的测试脚本，可以方便的收集测试结果。

6.4 本章小结

本章在现有的自动化系统之上，引入了新的观点。虽然这写观点，想法不一定很成熟，但是，都是从某个角度出发弥补现有系统的不足。如智能化测试平台的构建，针对当前的测试工具无法有效的将需求，测试用例，自动化脚本，缺陷有效的管理起来，提出了利用关系数据库模型将这些数据管理起来的新方法。又比如基于模型的测试方法，针对当前自动化脚本过于依赖数据的多元化而忽略了测试流程的多元化，提出了基于状态转换模型的测试流程。最后提出的持续集成

中的自动化回归测试，更是将STAF模型充分利用起来，实现了持续集成中的持续自动化回归测试，非常有希望成为持续集成技术中的一个流行技术。

第7章 总结

本文在软件测试行业自动化盛行的大背景下，针对该行业的自动化测试技术，分析和阐述了多个自动化测试框架的系统。并且提出了新的测试框架所具有的特点和能力。针对不同的软件自动化技术，分别剖析了各自的优势和不足，尤其是在实践中的使用价值，和具有针对性的设计模式。在实践中，通过使用这些有效的模式大大提高了测试工作效率，实现了简单有效的自动化回归测试方案。

本文第二章主要介绍自动化测试的发展，以及自动化测试系统在软件行业应用的背景介绍。具体研究工作主要集中在第三，四，五章。笔者分析了多种主流自动化系统的架构设计，阐述了各种系统的主要原理和优缺点。接着细化分析了某几个系统的核心模块——自动化测试框架引擎。给出了主要框架的详细设计思路，原理和功能流程。并且介绍了这些框架使用时的问题，并给出了多个已被成功实施的解决方案。

本文通过对软件自动化测试框架的分析，阐明了在金融行业实施自动化测试的必要性和可行性。最后，本文创新的提出了实现框架文档智能化管理，分模块执行，以及持续构建自动化的构思，并设计了初步的结合持续构建的系统整体构架。

笔者认为自动化软件测试在金融行业的广泛运用是大势所趋。下一步的工作主要针对提出的先进的模型来不断提高自动化软件测试的效率和可维护性，使得自动化真正能辅助软件工程师有效控制软件质量持续提升。

参考文献

[1] JORGENSEN, PAUL C. 软件测试. 韩柯，杜旭涛，译. 北京：机械工业出版社，2004.

[2] MARK FEWSTER, DOROTHY GRAHAM. 软件测试自动化技术与实例详解. 北京：电子工业出版社，2000.

[3] 郑人杰. 计算机软件测试技术. 北京：清华大学出版社，1990.

[4] 寇雅楠，李增智，王建国，等. 计算机软件测试研究. 计算机工程与应用，2002，(10)：27-31.

[5] 童荣胜，古天龙. 计算机科学与技术方法论. 北京：人民邮电出版社，2002.

[6] Rational. Principles of Software Testing for Testers, TST170, 2006

[7] Rational. Essentials of Test Management with Rational TestManager. TST275, 2006

[8] Rational.Mastering the Management of Iterative Development v2003.06.00. PRJ480, 2006

[9] Fewster, Graham. Software Test Automation. McGraw Hill, 2004

[10] Michael Kelly. Choosing a Test Automation Framework. IBM Developer Works. Senior SQA Specialist, CTI Group, 2005.

[11] Keith Zambelich. Totally Data-Driven Automation Testing. Automated Testing Specialists, Inc. 2006

[12] Carl Nagle. Test Automation Frameworks. SAFSDEV Web site. 2007

[13] 郦萌，王铁江.基于关键字驱动脚本的安全软件自动测试系统，同济大学学报（自然科学版）. 2002.

[14] Danny R. Faught. Keyword Driven Testing. Open Testware Reviews. 2006

[15] 侯勇 ，张海林. 自动化测试中的关键字驱动脚本技术 The Keyword Driven Script Technology in Automation Testing. 电子科技 IT Age. 2004

[16] Christian Hellsten. Using selenium with automation regression

test. IBM Developer Works. 2006

[17] Stephan Wiesner. Test-first development with FitNesse. JavaWorld.com. 2006

[18] 徐振良，樊滨温，王志鹏 关键字驱动技术在 SAFS 中的研究 The Study of Keyword Driven Technology in SAFS. 微计算机信息 Control & Automation. 2005

[19] 王莉，殷锋，李奇 软件自动化测试脚本设计研究 Research on the script designing for the software test automation. 西南民族大学学报（自然科学版）. 2003

[20] 冯玉才，唐艳，周淳 关键字驱动自动化测试的原理和实现 Theory and implementation of keyword-driven test automation. 计算机应用 Journal of Computer Applications. 2006

[21] 崔俊涛, 费伽, 陈颉.使用 CruiseControl 和 STAF 建立复杂环境下的编译和测试自动化. IBM Developer Works, 2008

作者简介

陆栋，男，生于1982年10月，籍贯浙江杭州。2001年至2005年在浙江大学就读计算机应用与工程本科并获得计算机学士学位。从2006年起攻读浙江大学计算机应用硕士学位。

本人于2005年5月加入浙江大学道富技术中心从事软件质量保证方面的工作。主要的研究方向是金融软件自动化测试技术，软件测试流程控制与管理，软件工程管理与项目估算等。曾在道富浙江首届软件质量保证研讨会上发表过关于缺陷管理技术的论文。目前的主要工作是完善软件自动化测试框架，并致力于推广自动化测试技术在金融软件测试领域的应用。

自动化软件测试框架分析及应用

作者： 陆栋
学位授予单位： 浙江大学计算机学院

万方数据 WANFANG DATA 文献链接

相似文献(10条)

1.学位论文 余琳 电网企业软件测试项目管理的研究 2006

随着经济的全球化，计算机及网络技术的快速发展，计算机软件行业与国际标准接轨已经是必然的发展结果。而计算机软件开发技术从个体开发、小团队开发到大型工程化开发的发展，已经是目前计算机软件业的一个基本趋势。计算机软件测试已经是计算机软件质量控制的主要手段。因此，各种软件测试工具、测试模型的涌现，使得软件测试技术的研究取得了很大的突破，计算机软件测试管理的作用日愈凸现。 本文结合作者的实际工作，对软件测试的发展历程和电网企业软件测试项目管理工作现状进行了系统研究。通过对比、实证、案例研究等方法，分析了计算机软件测试的发展历程，国内计算机软件行业存在的问题。特别是结合电网企业软件测试工作中存在的专业壁垒、需求变动大、测试管理方法相对缺乏等问题，本着理论联系实际的原则，在电网企业软件测试工作中提出项目管理的方法进行过程控制，并对软件测试项目管理的作用从软件项目的成本、工期、质量等方面进行了深入研究。并结合实际对电网企业软件测试项目管理方法进行了研究。 本文结合电网企业的特点以及软件测试项目管理的理论思想，对电网企业计算机软件测试项目管理的组织措施、技术保障以及质量体系建立等方面进行深入讨论，在此基础上，提出了一整套适合于电网企业软件测试项目管理的实施策略。

2.期刊论文 刘文红.王占武.马贤颖.LIU Wen-hong.WANG Zhan-wu.MA Xian-ying 基于CMM的软件测试过程管理 -现代计算机（专业版）2008(2)

针对软件测试项目特点,详细分析了软件测试过程的4个阶段,说明了各阶段应完成的主要任务和实施步骤,并结合CMM方法,提出了基于CMM的软件测试过程管理方法.阐述了该方法中软件测试需求管理、项目策划、项目监督与控制、质量保证和配置管理的内容和要求,实现了软件测试过程的规范管理,提高了测试过程的管理水平,保证了软件测试的质量与效率.

3.学位论文 雷欢 基于CMMI模型的H公司软件测试项目过程改进研究 2008

全球信息产业持续发展，并由硬件主导型向软件和服务主导型转变，软件质量是推动软件产业良性发展的核心。中国软件产业近年来飞速发展，而软件质量方面的投入却大大滞后于软件产业规模的增长。软件测试以发现软件中隐藏的缺陷为目的，贯穿软件开发生命周期的始终，是提高软件质量不可或缺的手段。H公司是H市唯一一家提供第三方软件专业测试的机构，公司已经成立两年多，此时的公司已经进入快速发展期，为了适应新的发展，公司的人员越来越多，规模迅速扩大，业务层次也在不断提高。与此同时，公司内部的项目管理也面临更大的挑战：缺乏对项目进度的监控，忽视对风险的管理和变更的控制等，这其中的任何一个问题如果没有得到很好的处理，就有可能导致项目的失败。 本研究按照CMMI的理论方法以及项目管理的基本思想，针对H公司软件专业测试项目管理中存在的不足，提出解决方案。因为CMMI模型是建立在软件开发项目的基础之上，所以本课题的创新点在于对H公司的软件测试项目，结合H公司的实际情况，对CMMI模型中的实践作相应的替代，使原本适用于软件开发行业的模型改进为适用于软件测试行业的框架。H公司由EPG（工程过程组）负责协调全组织软件测试过程的开发和改进活动。

4.期刊论文 刘文红.王占武.吴欣.张卫祥.陈青.LIU Wen-Hong.WANG Zhan-wu.WU Xin.ZHANG Wei-xiang.CHEN Qing 基于CMMI的软件测试项目过程管理 -飞行器测控学报2006,25(3)

针对独立的第三方软件测试项目特点,运用基于CMMI的软件项目管理方法,将需求管理、测试项目策划、项目监督与控制、质量保证和配置管理要求贯彻在软件测试过程中,给出了软件测试项目过程管理解决方案.该方案保证了软件测试项目的质量,提高了测试项目的技术水平和管理水平.

5.学位论文 吴永强 测试控制项目开发的方法研究——N模型 2007

本文以软件行业通用开发模型为切入点，简单描述瀑布开发模型、原型开发、螺旋模型、迭代模型和V模型。针对目前行业中普遍使用的V模型软件开发方法，通过具体项目实践，继承并进一步发展V模型为N模型；并从组织行为学和项目管理的角度阐述了N 模型的理论基础，结合V模型理论和项目管理知识重点论述了N 模型的过程、N 模型的意义和N 模型应用时的组织设计和人力资源管理问题。本文主要以理论加实践的方式，再结合自己在实践过程中的经验和体会，重点从测试角度来解决项目管理中的各开发阶段中的验收问题、进度控制问题、人员管理问题、项目的量化管理问题和项目质量问题。本着“任何事情都可能出错”以及“管理就是控制”的原则，本论文以V模型为依托，纵横比较，归纳总结，在新发展的N模型中融合了项目管理、组织行为学和人力资源管理等学科知识；可以说，比传统的软件开发模型纯粹以软件行业知识为基础又更进了一步。 采用N 模型以测试为核心管理和控制项目，使项目每一阶段结束的标准跟质量控制结合起来，将整个项目周期在不同的阶段建立里程碑，分阶段进行量化考核；不但注重进度，还保证各个里程碑的质量，从而保证了整个系统的质量，对项目进度、项目质量和人员绩效考核得到有效控制。N 模型将软件测试技术和项目管理有机结合起来，大大提高了软件测试在项目开发中的作用，又部分解决了软件开发行业的一些困境；对软件测试技术的发展起到一定推动作用，也拓展了软件项目管理的视野。尤其是在信息技术日新月异的今天，跨领域的知识交叉越来越多，本作者希望能通过此文不仅解决软件行业的一些问题，也希望有更多的人站在不同角度，以更开阔的视野，用各种跨领域的知识开拓和进一步发展N模型；这也是本论文的目的之一。

6.学位论文 韩霞 软件配置管理能力成熟度模型理论研究与应用 2007

随着科技的不断发展，软件项目的复杂性与集成度越来越高，软件过程当中产生的产品、文档也越来越多。对于软件测试来说，其变更比硬件、软件开发都要频繁的多，本文所在背景单位就是一个特殊的第三方软件测试机构，要适应测试当中纷繁复杂的变化，规范测试流程，就要在整个软件测试项目过程中实行项目管理，尤其是软件配置管理。 软件配置管理是美国国防部和卡耐基-梅隆大学的软件工程研究所发布的CMM(Capability Maturity Model)模型第二级的一个关键过程域(KPA)，该模型中从配置管理技术的角度说明了达到成熟度二级的企业应该做到哪些方面的工作。但是实际上配置管理不仅仅是一个技术的活动，还是一门管理的艺术。 从能力成熟度模型CMM到能力成熟度模型集成CMMI，都是保证软件质量和生产效率的重要手段，如何将CMM/CMMI应用于软件开发过程的各个阶段，一直是研究的热点。 本文以航天软件第三方测试机构为代表，针对此类专注于测试的测评机构进行测试过程中的软件配置管理活动，提供了理论依据与指导。结合CMM/CMMI模型提出了软件配置管理能力成熟度模型(SCM-CMM)，不但能够很好地解决配置管理的执行内容，还符合CMM的标准。SCM-CMM可以分为五个成熟度等级，包括混沌级、规范化级、已定义级、可度量级、持续优化级。其中每个等级分别有着自己不同的目标，是可以一级一级持续改进的。不同的企业可以根据自己需要裁剪符合自己要求的部分。 许多已经应用了CMM/CMMI的公司会发现基于CMM框架的SCM-CMM很容易上手。而且，SCM-CMM也能独立于CMM，用来评估、加强配置管理过程，没有实行过CMM/CMMI的企业也可以应用此模型为改进自己的软件配置管理过程能力作参考，所以SCM-CMM有着广泛的用途。 最后还给出了实例，给此类相似单位执行配置管理，提供了很好的参考。

7.期刊论文 梁成才.LIANG Chengcai 软件测评实验室软件测试项目的度量研究 -计算机工程2005,31(23)

对软件测评实验室承担的软件测试项目所需的度量进行了综述、分类和研究,列举了常用的项目度量、过程度量和产品度量,给出了典型度量的使用实例.

8.学位论文 何刚勇 客户驱动的项目管理方法在软件质量控制中的应用与探讨 2003

随着计算机技术的广泛应用，人们对软件产品的需求也日益增长。软件产品的特殊性决定了软件质量是软件产业发展关键问题。最初软件产品的质量问题表现为程序设计技巧问题，然后演变成为软件工程技术问题，目前发展成为软件工程项目管理问题。它贯穿了软件产业发展的各个阶段。
因此，软件项目管理应该是围绕软件产品质量这一核心进行的。而衡量质量的标准是客户的全面满意。因此，应把客户的全面满意作为软件项目管理的宗旨。在当前的软件项目管理实践中，脱离客户全面满意这一中心，往往导致项目的失败。 客户驱动的项目管理方法(Customer-

DrivenProjectManagement，简称“CDPM”)是一种融合全面质量管理(TotalQualityManagement)思想和现代项目管理(ProjectManagement)方法的方法体系，其核心思想是追求客户的全面满意。　　本文尝试结合本人在新加坡的项目经验，通过将CDPM应用在一个软件开发质量控制项目(即软件测试项目)的具体案例中，解决该项目在管理中遇到的各种问题。本文还提供了一些软件测试项目的管理经验、过程规范及相关文档。

9.学位论文 李春 C语言软件测试环境——Tester　1.0的设计与实现 1996

Tester　1.0是作者设计并开发的软件测试环境,它主要适用于C语言程序的单元测试.在此测试工具中,应用了静态分析,逻辑覆盖,模块驱动等软件测试技术.并且还具有测试项目管理,生成测试用例等功能以及友好的用户界面.利用此软件测试环境,可以提高用C语言进行程序设计的软件的测试效率,减少测试人员的工作量,方便测试项目管理,并且可以直接获得直观的测试文档.该论文是该人在进行软件测试工具设计和开发的实践的基础上完成的.论文介绍了Tester1.0的总体设计以及函数驱动部分的设计,并且结合该人在实际开发过程中获取的经验教训,讨论了软件测试工具的设计与开发中的要点和一般思路.论文首先介绍了软件测试的基本概念、内容、重要性以及常用的软件测试技术,并且对软件测试工具及其发展作了总体介绍.然后基于该人的实践经验讨论了软件测试工具设计的一般思路.随后详细说明了Tester1.0的总体设计,并对测试项目管理、静态分析、覆盖分析、函数驱动、用户界面以及测试数据库管理等模块分别进行了详细的介绍.然后又对函数驱动部分(包括函数接口定义、函数驱动程序自动生成、测试用例的生成和执行)的设计作了详细说明.最后,对Tester1.0的发展和实用化改进提出了若干建议.

10.期刊论文 丁智勇 项目质量管理过程中的软件测试问题 -科技创新导报2009(2)

本文探讨了项目质量管理过程中有关软件测试的经验,包括测试的概念、目的和原则,如何组织和管理软件测试,以及测试中常见的问题分析等.本文主要目的是分享软件开发项目实际工作中的经验和成果.从而能够增进开发组和测试组之间的相互了解,使开发组和测试组的配合更加默契,保证软件产品质量目标的实现.

本文链接：http://d.g.wanfangdata.com.cn/Thesis_Y1399719.aspx

下载时间：2010年4月21日